Современный
Гуманитарный
Университет

Дистанционное образование

1000.02.01;1

Рабочий учебник

Фамилия, имя, отчество _____

Факультет _____

Номер контракта _____

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

ЮНИТА 2

ОСНОВЫ ТЕОРИИ АЛГОРИТМОВ И ФОРМАЛЬНЫХ СИСТЕМ

320-200

53-39-90

продавец в окс-т по прод-же комп. тех.

Москва 2002

Разработано В.М. Кустовым

Рекомендовано Министерством образования Российской Федерации в качестве учебного пособия для студентов высших учебных заведений

КУРС: МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

Юнита 1. Элементы математической логики.
Юнита 2. Основы теории алгоритмов и формальных систем.

Юнита 2

Рассматриваются формальные системы, различные алгоритмические модели, рекурсивные множества и функции, теорема Геделя о неполноте математики, сложность вычислений.

Для студентов Современного Гуманитарного Университета

Юнита соответствует профессиональной образовательной программе № 1

© СОВРЕМЕННЫЙ ГУМАНИТАРНЫЙ УНИВЕРСИТЕТ, 2002

ОГЛАВЛЕНИЕ

ДИДАКТИЧЕСКИЙ ПЛАН 5
ЛИТЕРАТУРА 6
ПЕРЕЧЕНЬ УМЕНИЙ 7
ТЕМАТИЧЕСКИЙ ОБЗОР 10
1. Введение в формальные системы 10
1.1. Аксиоматический метод 10
1.2. Понятие о метаязыке и метатеории 16
1.3. Интерпретация формальной системы и теории 20
1.4. Структура языка и выражения. Функторы 22
1.5. Грамматики 25
1.6. Исчисление высказываний: интуитивный подход 27
1.7. Исчисление высказываний: формальный подход 34
1.8. Определение формальной системы 39
2. Алгоритмы как формальные системы 41
2.1. Интуитивное понятие алгоритма 41
2.2. Формализация и обобщение понятия алгоритма 46
2.3. Марковские алгоритмы 47
2.4. Челночные алгоритмы 52
2.5. Вычислимые функции 55
3. Машина Тьюринга и вычислимость 59
3.1. Основы теории формальных грамматик 59
3.2. Регулярные языки и автоматные грамматики 61
3.3. Конечные автоматы 65
3.4. Машина Тьюринга 70
3.5. Вычислимость по Тьюрингу 81
3.6. Тезис Черча 87
4. Рекурсивные множества и функции 92
4.1. Понятие рекурсии 92
4.2. Рекурсивные и рекурсивно перечислимые множества и предикаты 94
4.3. Примитивно рекурсивные функции 108
4.4. Теорема о существовании универсальной частично рекурсивной функции 118
4.5. Формальная арифметика 122
4.6. Теорема Геделя о неполноте математики 125
5. Сложность вычислений и элементы логического программирования 135
5.1. Меры сложности вычислений 135

5.2. Формальные языки класса *P* 138
5.3. Недетерминированная машина Тьюринга и язык *NP* 140
5.4. Понятие *NP*-полной задачи 143
5.5. Хорновские дизъюнкты 145
5.6. Логические программы 148
ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ 150
ТРЕНИНГ УМЕНИЙ 155
ФАЙЛ МАТЕРИАЛОВ 174
ГЛОССАРИЙ*

* Глоссарий расположен в середине учебного пособия и предназначен для самостоятельного заучивания новых понятий.

ДИДАКТИЧЕСКИЙ ПЛАН

Введение в формальные системы. Аксиоматический метод. Понятие о метаязыке и метатеории. Интерпретация формальной системы и теории. Структура языка и выражения. Функторы. Грамматики. Исчисление высказываний: интуитивный подход. Исчисление высказываний: формальный подход. Определение формальной системы.

Алгоритмы как формальные системы. Интуитивное понятие алгоритма. Формализация и обобщение понятия алгоритма. Марковские алгоритмы. Челночные алгоритмы. Вычислимые функции.

Машина Тьюринга и вычислимость. Основы теории формальных грамматик. Регулярные языки и автоматные грамматики. Конечные автоматы. Машина Тьюринга. Вычислимость по Тьюрингу. Тезис Черча.

Рекурсивные множества и функции. Понятие рекурсии. Рекурсивные и рекурсивно-перечислимые множества и предикаты. Примитивно рекурсивные функции. Теорема о существовании универсальной частично рекурсивной функции. Формальная арифметика. Теорема Геделя о неполноте математики.

Сложность вычислений и элементы логического программирования. Меры сложности вычислений. Формальные языки класса *P*. Недетерминированная машина Тьюринга и язык *NP*. Понятие *NP*-полной задачи. Хорновские дизъюнкты. Логические программы.

ЛИТЕРАТУРА

Базовая

*1. Гросс М., Лантен А. Теория формальных грамматик. М., 1971.

*2. Гладкий А. Математическая логика. М., 1998.

*3. Кагленд Н. Вычислимость. Введение в теорию рекурсивных функций. М., 1983.

Дополнительная

4. Верещагин Н., Шень А. Вычислимые функции: Лекции по математической логике и теории алгоритмов. М., 1999.

*5. Карри Х. Основания математической логики. М., 1969.

*6. Клини С. Математическая логика. М., 1973.

*7. Мендельсон Э. Введение в математическую логику. М., 1971.

8. Носов В. Основы теории алгоритмов и анализа их сложности. М., 1992.

*9. Роджерс Х. Теория рекурсивных функций и эффективная вычислимость. М., 1972.

10. Справочная книга по математической логике. В 4-х ч. Ч. 1. Теория моделей / Под ред. Барвайса. М., 1982.

Примечание. Знаком (*) отмечены работы, на основе которых составлен тематический обзор.

ПЕРЕЧЕНЬ УМЕНИЙ

№ п/п	Умения	Алгоритмы
1	Нахождение функции $h(m,k)$ в рекуррентной формуле $f(n+1)=h\big(n,f(n)\big)$ для одноместной функции $f(n)$	1. Подставить в функцию $f(n)$ вместо n значение $n+1$. 2. Выразить $f(n+1)$ через $f(n)$ и n. 3. Найти функцию $h(m,k)$ из формулы $f\big(n+1\big)=h\big(f(n),n\big)$, заменив $f(n)$ на m, а n – на k.
2	Нахождение функций g и h в рекурсивной формуле для двухместной функции	1. Найти функцию g: $g(x)=f(x.0)$ (если рекурсия проводится по y). 2. Подставить в функцию $f(x,y)$ вместо y значение $y+1$. 3. Выразить функцию $f(x,y+1)$ через $f(x,y)$, x, y. 4. Найти функцию $h(x,y,z)$ из формулы $f(x,y+1)=h\big(x,y,f(x,y)\big)$, заменив $f(x,y)$ на z. *Замечание.* Если рекурсия проводится по x, то $g(y)=f(0,y)$, а $h(x,y,z)$ находится из формулы $f(x+1,y)=h\big(x,y,f(x,y)\big)$.
3	Нахождение функций g и h в рекурсивной формуле для трехместной функции	1. Найти функцию g: $g(x,y)=f(x,y,0)$ (если рекурсия проводится по z). 2. Подставить в функцию $f(x,y.z)$ вместо z значение $z+1$. 3. Выразить функцию $f(x,y,z+1)$ через $f(x,y,z)$, x, y, z. 4. Найти функцию $h(x,y,z,t)$ из формулы $f(x,y,z+1)=h\big(x,y,z,f(x,y,z)\big)$, заменив $f(x,y,z)$ на t. *Замечание.* То же самое проделывается для других переменных, если по ним проводится рекурсия (см. замечание к умению 2).
4	Нахождение рекурсивных формул для двухместной функции $f(x,y)$	1. С помощью умения 2 найти функции g и h (если рекурсия проводится по y). 2. Записать рекурсивные формулы, используя соотношения $\begin{cases} f(x,0)=g(x) \\ f(x,y+1)=h\big(x,y,z\big)\big\|_{z=f(x,y)} \end{cases}$ (если рекурсия проводится по y – см. замечание к умению 2).

5	Нахождение рекурсивных формул для трехместной функции $f(x,y,z)$	1. С помощью умения 3 найти функции g и h (если рекурсия проводится по z). 2. Записать рекурсивные формулы, используя соотношения $\begin{cases} f(x,y,0)=g(x,y) \\ f(x,y,z+1)=h\big(x,y,z,t\big)\big\|_{t=f(x,y,z)} \end{cases}$ (если рекурсия проводится по y – см. замечание к умению 3).
6	Доказательство принадлежности двухместной функции классу п.р.ф.	1. С помощью умения 2 найти функции g и h. 2. Выразить $g(x)$ через исходные п.р.ф. (если рекурсия проводится по y). 3. Выразить $h\big(x,y.z\big)$ через исходные п.р.ф. 4. Сделать вывод о принадлежности функции классу п.р.ф.: если функция $f(x,y)$ может быть получена из исходных п.р.ф с помощью операторов подстановки и рекурсии, то она принадлежит классу п.р.ф. *Замечание.* То же самое проделывается для x, если по нему проводится рекурсия (см. замечание к умению 2).
7	Нахождение геделевского номера ng одноместной п.р.ф.	1. Найти число c по формуле $c=g(0)=f(0)$. 2. Подставить в функцию $f(x)$ вместо x значение $x+1$. 3. Выразить функцию $f(x+1)$ через $f(x)$ и x. 4. Найти функцию $h(x,y)$ из формулы $f(x+1)=h\big(x,f(x)\big)$, заменив $f(x)$ на y. 5. Найти геделевский номер j функции h. 6. Найти геделевский номер N функции $f(x)$ по формуле $ng=2^{c+1}\cdot 5^{j}$, где $c=g(0)$.
8	Нахождение геделевского номера двухместной п.р.ф.	1. Найти функцию $g(x)=f(x,0)$. 2. Найти функцию $h(x,y,z)$ из уравнения $f(x,y+1)=h\big(x,y,f(x,y)\big)$ (см. умение 2). 3. Найти геделевский номер i функции $g(x)$ (см. умение 7). 4. Найти геделевский номер j функции $h(x,y,z)$. 5. Найти геделевский номер ng двухместной функции $f(x,y)$ по формуле $ng\big(f(x,y)\big)=3^{i}\cdot 5^{j}$.

9	Нахождение геделевского номера трехместной п.р.ф.	1. Найти функцию $g(x,y)=f(x,y,0)$. 2. Найти функцию $h(x,y,z,t)$ из уравнения $f(x,y,z+1)=h\big(x,y,z,f(x,y)\big)$ (см. умение 2). 3. Найти геделевский номер i функции $g(x,y)$ (см. умение 7). 4. Найти геделевский номер j функции $h(x,y,z,t)$. 5. Найти геделевский номер ng двухместной функции $f(x,y,z)$ по формуле $ng\big(f(x,y,z)\big)=3^i\cdot 5^j$.
10	Расшифровка геделевского номера	1. Если число n простое, то $f=e_k^1$. 2. Если число $n=2^{c+1}\cdot 5^j$, то: если j – номер какой-либо двухместной п.р.ф., то n – номер одноместной п.р.ф, получающейся из c и h с помощью оператора рекурсии; если j – не номер п.р.ф, то n – не номер п.р.ф. 3. Если $n\neq 2^{c+1}\cdot 5^j$, то проверяем, равно ли $n=3^i\cdot 5^j$; если да, то i и j – номера п.р.ф. вместимости $k-1$ и $k+1$ соответственно. И т.д.

1. ВВЕДЕНИЕ В ФОРМАЛЬНЫЕ СИСТЕМЫ

1.1. Аксиоматический метод

Исторически понятие формальной системы развилось на основе понятия аксиоматической теории.

Аксиоматический метод – это способ построения научной теории, при котором в основу теории кладутся некоторые исходные положения, называемые аксиомами, а все остальные положения теории (вспомогательные – леммы и ключевые теоремы) получаются как логические следствия аксиом.

Внутренняя логическая связность некоторой математической теории и ее адекватность для описания того или иного круга физических явлений – это две совершенно разные вещи. Данный факт был осознан математиками, когда им пришлось объяснять возможность одновременного существования евклидовой геометрии и различных неевклидовых геометрий.

Чтобы построить и логически развить некоторую математическую теорию, необходимо абстрагироваться от природы объектов, которые она рассматривает (вернее, от природы объектов, которые как бы мотивировали извне создание этой теории). В то же время необходимо самым тщательным образом уточнить с помощью аксиом основные отношения, связывающие эти объекты друг с другом.

Аксиоматико-дедуктивный метод в математике приобрел известность благодаря "Началам" Евклида (появившимся около 330–320 гг. до н.э.), но традиция приписывает открытие этого метода Пифагору (VI в. до н.э.). С помощью аксиоматического метода была систематизирована совокупность геометрических знаний. Евклидову аксиоматическую систему в общих словах можно охарактеризовать следующим образом. Даются определения некоторых первоначальных (исходных) терминов, таких как точка, прямая, плоскость; определения эти преследуют цель объяснить значения данных терминов. Затем в качестве аксиом (постулатов) принимаются некоторые предложения об этих первоначальных терминах; имеется в виду, что эти предложения непосредственно очевидны на основе понимания первоначальных терминов, которое подсказывается определениями. Затем через первоначальные термины определяются новые термины, а из аксиом

* Жирным шрифтом выделены новые понятия, которые необходимо усвоить. Знание этих понятий будет проверяться при тестировании.

логически выводятся новые предложения, называемые теоремами. Аксиоматику, подобную евклидовой, где значения исходных терминов предполагаются данными с самого начала, называют материальной (содержательной, неформальной) аксиоматикой.

Один из постулатов Евклида – пятый постулат, или "постулат о параллельных", – кажется менее очевидным, нежели остальные. Евклид использовал этот постулат для доказательства теоремы о том, что через данную точку P, не лежащую на данной прямой l, можно провести в точности одну прямую, параллельную l (т. е. не пересекающую ее ни в какой точке). Начиная со времен самого Евклида было предпринято множество попыток доказать этот постулат как теорему, выведя его из остальных постулатов евклидовой системы. Как мы знаем теперь, попытки эти были обречены на неудачу.

Дело в том, что Лобачевский в 1829 г. и Бойаи в 1833 г. построили геометрическую систему, в которой через данную точку P, не лежащую на данной прямой l, можно провести бесконечно много прямых, параллельных l. Очевидно, что осмысление первоначальных терминов евклидовой геометрии на языке понятий физического пространства недостаточно для решения вопроса о том, какой же из постулатов о параллельных верен: постулат Евклида или постулат Лобачевского – Бойаи. Различия в получающихся при этом геометрических системах могут быть слишком малыми для того, чтобы обнаружить их посредством каких бы то ни было измерений в доступной нам части Вселенной – точно так же, как в былые времена люди полагали, что Земля плоская на основании наблюдений над доступной непосредственному обозрению частью земной поверхности.

Поэтому истинность какого-либо предложения евклидовой геометрии должна быть свойством самой этой геометрии как логической системы. Но если евклидова геометрия – это корректная логическая структура, то это же можно сказать и о геометрии Лобачевского – Бойаи. В самом деле, как показал в 1871 г. Клейн, все аксиомы планиметрии Лобачевского оказываются истинными, если входящие в них первоначальные термины переинтерпретировать таким образом, чтобы плоскость понималась как внутренность некоторого круга в евклидовой плоскости, точка – как точка внутри этого круга, прямая – как хорда его окружности, а расстояния и углы вычислялись согласно формулам, предложенным в 1859 г. Кэли. (Другая евклидова модель, пригодная для интерпретации ограниченной части неевклидовой плоскости, была предложена в 1868 г. Бельтрами, который интерпретировал отрезки

прямых как отрезки "геодезических" (т.е. кратчайших путей, соединяющих две точки) на некоторой "поверхности постоянной отрицательной кривизны".)

В этих моделях с аксиомами происходит нечто новое по сравнению с прежними аксиоматическими концепциями: смысл первоначальных терминов варьируется, а дедуктивная структура рассматриваемой теории остается фиксированной. Это обстоятельство знаменует возникновение формальной аксиоматики, в рамках которой значения первоначальных терминов не предполагаются определенными с самого начала, а так и остаются неопределенными при выводе теорем из аксиом. Поэтому мы вольны выбирать значения этих первоначальных терминов любым образом, лишь бы аксиомы оставались истинными. Особенно плодотворной эта точка зрения оказывается в современной алгебре для получения следствий из рассматриваемых чисто формально систем аксиом, скажем, из аксиом абстрактной теории групп. Результаты, выводимые из аксиом теории групп, в которых множество элементов, на котором задана групповая операция умножения, и сама эта операция остаются неопределенными (нефиксированными), составляют теоретическую систему, пригодную для самых различных приложений.

В рамках формальной аксиоматики система аксиом может быть исследована на предмет наличия таких свойств, как независимость какой-либо аксиомы от других (посредством попыток отыскания такой интерпретации ее первоначальных терминов, при которой данная аксиома оказывалась бы ложной, а все остальные – истинными) и категоричность (заключающаяся в том, что элементы двух произвольных интерпретаций можно поставить во взаимно однозначное соответствие с сохранением всех определенных для них свойств).

При таком подходе к аксиоматике возникает ряд вопросов. Почему мы выбираем именно эти аксиомы и почему получающаяся в результате система должна нас интересовать? Ответ, очевидно, состоит в том, что такая система приложима к любым объектам, заданным извне в качестве интерпретации первоначальных терминов. Бывает, что система аксиом допускает существенно различные интерпретации (в этом случае она некатегорична); типичным примером могут служить аксиомы абстрактной теории групп. Мы не хотели бы рассматривать системы аксиом, которым не удовлетворяет никакая интерпретация; такие системы мы назовем вырожденными. Одна из проблем, встающих в формальной аксиоматике, как раз и состоит в установлении того, что данная система не является вырожденной. Впрочем, система объектов, служащая интерпретацией какой-либо системы аксиом, часто берется из какой-нибудь другой аксиоматической теории; в этом случае налицо некоторая редукция:

стоявший перед нами вопрос лишь заменяется вопросом о значении этой другой аксиоматической теории. Если ни на какой стадии процесса результаты не прилагаются за пределами формальной аксиоматики, то все эти построения кажутся бесплодными. Поэтому, если не становиться на позицию математического нигилизма, мы приходим к выводу, что формально аксиоматизированная математика – это еще не вся математика: определенное место в математике должны занимать такие понятия, как "смысл", "истина", "ложь". Когда мы утверждаем, что такое-то предложение данной формальной аксиоматической теории является теоремой, то мы, во всяком случае, должны считать само наше утверждение истинным в том смысле, что предложение, о котором идет речь, вытекает из аксиом, хотя вопрос о том, истинно ли это предложение в действительности, остается открытым, поскольку в формальной аксиоматике формальные выводы проводятся до какого бы то ни было приписывания значений первоначальным терминам (или безотносительно к такому приписыванию).

В качестве примера математического высказывания, относительно которого отнюдь не предполагается, что оно является чисто формальным, но бессмысленным следствием из аксиом, рассмотрим теорему (доказываемую в теории чисел) о том, что для любых данных целых чисел a, b, c мы можем узнать, существуют ли такие целые числа x и y, что $ax+by+c=0$, т. е. теорему о существовании общего метода, позволяющего решить вопрос, разрешимо ли в целых числах произвольное данное уравнение $ax+by+c=0$ (a, b, c – целые). Хотя теория чисел (арифметика целых чисел) может быть построена аксиоматически, доказательство этой теоремы следует понимать в том смысле, что для любых конкретных a, b, c мы можем узнать, разрешимо ли данное уравнение. Ученик, научившийся лишь доказывать, исходя из аксиом, теорему о том, что всегда можно узнать, существует ли искомое решение, но не научившийся фактически узнавать это, вряд ли усвоил то, чему хотел его научить учитель. В то же время если имелся в виду лишь вывод этой теоремы в формальной аксиоматике, то такой ученик сделает все, что от него требуется.

Один из наиболее скромных методов разрешения ситуации, сложившейся в связи с парадоксами, – использование некоторой системы аксиоматической теории множеств. В этом контексте аксиоматика должна пониматься в формальном смысле, если только мы не станем пытаться каким-либо образом сохранить как раз ту интуитивную концепцию множества, которая должна была быть

заменена именно этой системой аксиом. Однако рассмотренное выше показывает, что "прибежище", предоставляемое формальной аксиоматикой, хотя и может обеспечить значительные удобства, оставляет открытыми такие проблемы, как выбор аксиом и приложимость их к описанию каких-либо систем объектов, кроме таких, существование которых постулировано какой-нибудь другой аксиоматической теорией.

За решение такого рода проблем взялся Гильберт. Он исходил из того, что классическая математика (т.е. обычная математика, использующая классическую логику) содержит много такого, что выходит за рамки непосредственно осмысливаемого и обосновываемого на интуитивной основе. И действительно, математикам пришлось осознать это, когда в теории множеств они зашли слишком далеко и натолкнулись на парадоксы. Гильберт предложил программу спасения классической математики, ставящую своей целью избавление ее (математики) от парадоксов. Программа эта в общих чертах описывается следующим образом. Классическая математика должна быть сформулирована в виде формальной аксиоматической теории, после чего следует доказать ее непротиворечивость, т. е. установить, что в этой формальной аксиоматической теории нельзя доказать противоречие.

До этого предложения (оно было впервые высказано в 1904 г., но Гильберт и его сотрудники по-настоящему занялись им лишь к 1920 г.) доказательства непротиворечивости формальных аксиоматических теорий проводились с помощью построения моделей (интерпретаций), в которых все аксиомы данной теории оказывались истинными, когда входящие в них первоначальные термины интерпретировались посредством некоторой другой теории. Выше мы приводили один пример такого рода, который показывает, что неевклидова планиметрия Лобачевского непротиворечива, если непротиворечива планиметрия Евклида. В каждом таком случае доказательство непротиворечивости с помощью модели показывает лишь, что данная теория непротиворечива, если непротиворечива некоторая другая теория. Аналитическая геометрия Декарта (1619 г.) легко сводит вопрос о непротиворечивости геометрии к вопросу о непротиворечивости теории действительных чисел, т. е. к вопросу о непротиворечивости анализа. Но как установить непротиворечивость анализа? Разумеется, не при помощи геометрической модели: это привело бы к порочному кругу. Согласно Гильберту и Бернайсу, для этой цели нельзя обратиться и к физическому миру. Дело в том, что ограниченная точность физических измерений не позволяет нам заявить, что континуум действительно дан нам в опыте; правильнее будет сказать, что континуум – это идея, полученная путем

экстраполяции (или идеализации) того, что действительно является данными опыта.

В силу сказанного предложение Гильберта доказать непротиворечивость классической математики, построенной в виде формальной системы, непременно предполагало использование некоторого нового метода взамен метода построения модели. Этот метод состоит в непосредственном использовании идеи непротиворечивости, сводящейся к отсутствию противоречия (парадокса), т. е. двух теорем, являющихся отрицаниями друг друга. Чтобы показать, что такой ситуации возникнуть не может, Гильберт предложил сделать доказательства в аксиоматической теории предметом специальной математической дисциплины, названной им метаматематикой, или теорией доказательств. Конечно, убедительность такого доказательства непротиворечивости должна зависеть от используемых в метаматематике методов. По этой причине Гильберт решил использовать в метаматематике лишь интуитивно убедительные методы, которые он называл "финитными". В особенности такие методы должны избегать использования "актуальной" ("завершенной") бесконечности. Новый подход Гильберта позволяет избежать использования актуальной бесконечности и в самой формулировке проблемы доказательства непротиворечивости. Дело в том, что в любой данной теории имеется лишь счетно-бесконечное множество доказательств, а в утверждении о ее непротиворечивости говорится лишь о произвольной паре доказательств, а не обо всем множестве доказательств как о завершенном объекте. Предполагаемые объекты рассмотрения самой теории могут быть значительно менее элементарными. Таким образом, казалось правдоподобным, что проблема непротиворечивости, сформулированная в финитных терминах, может быть и решена финитными методами.

Ниже мы подробно обсудим, каким образом некоторые разделы математики могут быть представлены в виде формальных аксиоматических теорий и изучены в метаматематике.

Определение 1.1. **Аксиомой** называется исходное положение научной теории, принимаемое без доказательства (от греч. axioma – принятие положения).

Каким именно способом выводятся теоремы из аксиом в рамках аксиоматической теории, об этом в большинстве случаев специально не говорят. Считается, что это делается в соответствии с правилами логики, которые основаны на здравом смысле и усваиваются на примерах. Эти правила к тому же представляются "очевидными", например: если некоторое высказывание истинно; то его отрицание ложно.

Долгое время, по крайней мере во Франции, исследование логических правил было предоставлено "логикам", которые обычно не знали математики (и работы которых считались бесперспективными).

Эта ситуация изменилась, как только математики стали заниматься математическими объектами, столь малоинтуитивными по своему характеру, что для их понимания "логики здравого смысла" оказалось недостаточно.

Переход от "наивных аксиоматических теорий" к "формальным системам" связан с осознанием того факта, что само понятие "логических правил" содержит в себе нечто неясное и потому нуждается в уточнении. Но тогда исследователь оказывается в ситуации, которая аналогична ситуации на уровне аксиом: мы по своему произволу выбирали аксиому Евклида или аксиому Лобачевского; точно так же мы можем по своему произволу выбирать одни или другие правила вывода и получать тем самым разные формальные системы.

1.2. Понятие о метаязыке и метатеории

В подразд. 1.1 при обсуждении проблемы формальной аксиоматики мы подчеркивали, что при выводе теорем из аксиом первоначальные термины рассматриваются как лишенные смысла, либо им вообще не приписывается никакого значения, либо же значение, которое они имеют, не принимается во внимание. Сказать же, что они обладают некоторым смыслом, играющим роль в процессе доказательства теорем, значит сказать, что для этих теорем важны некоторые свойства первоначальных терминов, помимо тех, что фигурируют в аксиомах. Но тогда описания этих дополнительных свойств следует сформулировать как дополнительные аксиомы.

Евклиду в "Началах" не удалось описать в его "аксиомах и постулатах" все свойства, используемые им в действительности. Его доказательства сопровождаются многочисленными чертежами. И понадобилось немало времени, чтобы прийти к очевидной для нас мысли, что на самом деле чертежи не должны быть существенной частью самого процесса доказательства: они либо облегчают процесс поиска доказательства, либо помогают следить за ходом доказательства, либо, наконец, способствуют запоминанию доказательств. На практике, однако, чертежи иногда служат источником информации, существенной для самого доказательства.

Последнее обстоятельство можно проиллюстрировать на печальном примере "теоремы", в которой "доказывается", что каждый треугольник – равнобедренный. От многих доказательств Евклида такие

"доказательства" отличаются лишь использованием чуть искаженных чертежей.

Используемые Евклидом "скрытые" допущения были выявлены уже в новейшее время и зафиксированы в виде аксиом Пашем, Гильбертом и другими учеными. Скажем, среди аксиом системы Гильберта имеется следующая (сформулированная впервые Пашем): если прямая, лежащая в плоскости некоторого треугольника и пересекающая одну из его сторон, не проходит через вершину треугольника, противолежащую данной стороне, то она пересекает одну из остальных сторон этого треугольника. На наших чертежах это так, но в тексте Евклида нет ничего, что позволило бы доказать, что это обязательно должно быть так. Изящное изложение евклидовой геометрии средствами формальной аксиоматики с явной формулировкой всех принимаемых допущений дано Гильбертом в его "Основаниях геометрии".

Обратившись снова к формальной аксиоматике, мы видим, что, кроме первоначальных терминов, самих по себе смысла не имеющих, в проведении выводов используется еще смысл слов естественного языка. Но мы уже видели, что различные теории могут отличаться друг от друга не только принятыми в них математическими допущениями, но и логическими средствами. Поэтому, чтобы понятие теоремы данной теории стало совершенно точным, нам придется в качестве следующего шага проделать процедуру, примененную ранее к первоначальным терминам, и по отношению ко всем словам, используемым в доказательствах. Иными словами, проводя точку зрения формальной аксиоматики, мы должны отвлечься от смысла всех входящих в доказательства слов и делать выводы исключительно на основе точно сформулированных правил, относящихся лишь к форме (а не к содержанию) предложений. Логика, используемая в процессе вывода в формальной аксиоматике, как раз и должна быть зафиксирована в виде таких правил, по крайней мере частично; частично же эта логика может быть выражена посредством логических аксиом.

Такая полная формализация была бы неосуществимой, если бы формализуемая теория формулировалась на естественном языке (скажем, на русском или английском): нерегулярности и неоднозначности, присущие словам и выражениям естественных языков, сильно осложнили бы эту задачу.

Фактически современная математика с большой выгодой пользуется специальными символизмами, но, как правило, часть ее предложений, в том числе используемых в процессе логического вывода, выражена обычными языковыми средствами. Символическая запись равенств не только весьма экономна, но и представляет

значительные удобства для различных преобразований (скажем, из $x+5=2$ получаем $x=2-5$, а затем и $x=-3$), которые, хотя и имеют содержательное обоснование, но на практике обычно производятся быстро, без задержек на обдумывание и обоснование каждого шага. Фактически это полуформальный способ рассуждений, в значительной мере обусловливающий мощь современной математики.

Полная формализация, которую мы собираемся провести, чтобы осуществить замыслы Гильберта и достичь некоторых других целей, получается путем сочетания обычной символики, принятой в современной математике, с символическим построением логики, идущим от работ Буля, Пирса, Фреге, Уайтхеда и Рассела и других авторов. Пользуясь обеими этими идеями, мы строим полностью символический язык для формализуемой теории. Для этого языка мы точно определяем синтаксис (посредством "правил образования") и логику (посредством "правил вывода", или "правил преобразования"). То, что получается в результате, мы будем называть формальной системой, или формализмом, или логистической системой. Сам этот метод, приводящий к уточнению понятия теории, иногда называют логистическим методом.

Обсуждение свойств некоторой формальной системы, в том числе точное ее описание (т.е. определение правил образования и преобразования) и исследование относящихся к ней результатов, мы производим в некоторой другой теории (средствами другого языка), которую назовем *метатеорией* (а этот другой язык – метаязыком). Саму же формальную систему мы будем называть предметной (или объектной) теорией (и соответственно употреблять термины "предметный язык" или "язык-объект").

Определение 1.2. **Метаязыком** называется язык, на котором описывается другой язык.

Замечание. Некоторые авторы придерживаются более узкого понимания термина "метаязык", ограничивая его финитными методами.

Мы рассматриваем изучаемый нами язык как "формальную систему", только если это символический язык (а не часть какого-либо естественного языка, скажем, русского или английского) и если для него точно определены аксиомы и правила вывода.

Изучение свойств формальной системы, проводимое содержательными математическими методами в рамках метаязыка, мы будем называть метаматематикой, или *теорией доказательств*. В современной науке термин "теория доказательств" используется в несколько более широком смысле, чем имел в виду Гильберт, согласно программе которого метаматематика должна пользоваться лишь так называемыми "финитными" методами, которые являются интуитивно

убедительными. Финитизм Гильберта представляет собой аксиоматический метод рассмотрения формальных теорий содержательной математики и исследование вопросов непротиворечивости таких теорий.

Метаязыковые символы, употребляемые в дальнейшем, будут определяться с помощью по возможности однозначных и логически отчетливых русских фраз, которые будут использоваться содержательным образом, т. е. на основе смысла слов, а не с помощью формальных правил (введение и употребление которых потребовало бы создания метаметаязыка). Таким образом, метаязыковые символы по существу представляют собой сокращения.

Пример 1.1. Пусть алфавит формальной системы есть

$$A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -\}.$$

Мы хотим определить два множества слов, а именно множество целых чисел без знака и множество целых чисел. Чтобы сформулировать правила образования этих множеств, мы введем метаязыковую переменную C, которая принимает любое значение из множества $\{0,1,...,9\}$, метаязыковую переменную W, принимающую значение из множества целых чисел без знака, и метаязыковую переменную E, принимающую значение из множества целых чисел.

Ломаные скобки "$\langle$" и "$\rangle$" будут использоваться для разграничения вхождений переменных; вертикальная черта "|" обозначает союз "или" в одном из его значений; наконец, символ "::=" означает разрешение заменить то, что стоит слева от него, каким-либо из выражений, стоящих справа от него и разделенных вертикальными чертами. Для обозначения конкатенации (соединения $A \cdot B$ двух цепочек A и B) используется "классический" типографский прием – простое соположение (без специального знака).

Приведем правила порождения интересующих нас множеств:

правило 1: $\langle W \rangle ::= \langle C \rangle | \langle W \rangle \langle C \rangle$;

правило 2: $\langle E \rangle ::= \langle W \rangle | +\langle W \rangle | -\langle W \rangle$.

Применяя правило 1, мы можем получить, например,

$$\langle W \rangle ::= 3$$
$$\langle W \rangle ::= 35,$$

применение правила 2 к последнему результату дает, например,

$$\langle E \rangle ::= -35.$$

Предположим, что алфавит некоторой формальной системы содержит букву " $\wedge$ " и имеется правило: "Если A и B – слова типа θ, то слово $A \cdot \wedge \cdot B$ – тоже слово типа θ". Русское выражение "если..., то..." представляется в данном контексте абсолютно однозначным, однако сугубо для краткости записи его можно заменить метаязыковым символом, например " $\Rightarrow$ ".

1.3. Интерпретация формальной системы и теории

Поскольку формальная система получается (как правило) в результате формализации некоторых разделов обычной неформальной или полуформальной математики, то символы, формулы и прочие элементы такой формальной системы истолковываются (интерпретируются) в терминах соответствующей неформальной или полуформальной математики.

Определение 1.3. **Интерпретацией теории** называется установление соответствия между элементарными высказываниями формальной теории и содержательными высказываниями некоторой предметной области.

Интерпретация теории является соответствием между высказываниями, при этом каждый из соответствующих друг другу членов является высказыванием с собственным критерием истинности. Другими словами, интерпретация – это совокупность значений, приписанных таким образом символам, формулам и прочим элементам формальной системы. Если мы не знаем этой интерпретации, данная формальная система не представляет для нас интереса. Если этой неформальной или полуформальной математической теории свойственна неопределенность, допускающая различные интерпретации (примером может служить абстрактная теория групп), мы можем употреблять термин "интерпретация" в единственном или во множественном числе в зависимости от того, на чем мы концентрируем наше внимание: на самой полуформальной математике как продукте творчества математиков или же на ее различных интерпретациях.

Говорят, что интерпретация теории полна, если каждому элементарному высказыванию теории соответствует некоторое содержательное высказывание; в противном случае интерпретация будет частичной.

В метаматематике, в соответствии с ее задачами, формальная система должна изучаться именно как таковая, т. е. просто как система символов, лишенных всякого смысла, а ее интерпретация не должна приниматься во внимание. Когда мы говорим об интерпретации, мы не занимаемся метаматематикой.

Обычно формальная система создается как идеализированный образ чего-то другого (это "другое" можно было бы назвать, например, "теорией"), по отношению к чему эта формальная система и обретает смысл. Тогда формальная система представляет собой формализацию некоторой теории, образуя синтаксис последней; теория же представляет собой интерпретацию данной формальной системы. Правильно построенные слова (правильные фразы) формальной системы получают смысл в рамках формализуемой теории; обратно, любому высказыванию, имеющему смысл в данной теории, соответствует некоторый класс правильно построенных слов (правильных фраз) в формальной системе.

Вопрос об адекватности данной формальной системы для данной теории – это проблема, относящаяся к методологии науки.

Исследование подобных проблем должно, разумеется, вестись как можно более строгими методами, но оно может выходить за рамки математики как таковой.

До сих пор мы шли от предметной области к теории и далее – к формальной системе. Обратное движение от формальной системы к некоторым объектам опыта приводит к понятию представления системы (рис. 1.1).

Рис. 1.1. Интерпретация и представление формальной системы

Определение 1.4. **Представлением системы** называется любой способ рассмотрения объектов формальных систем как конкретных объектов (полученных из опыта) при условии, что содержательные объекты сохраняют структуру формальных.

Всякая формальная система произвольного типа имеет синтаксическое представление.

1.4. Структура языка и выражения. Функторы

Прежде всего в нашем языке нам понадобятся однозначно построенные предложения, внутренняя структура которых нас совершенно не будет интересовать (пока мы остаемся в пределах исчисления высказываний), – нам надо лишь уметь распознавать и различать их. Мы назовем эти предложения элементарными формулами, или атомами. Различные буквы будут представлять различные атомы, а каждая из них на протяжении любого конкретного рассуждения должна обозначать один и тот же атом.

Кроме того, нам будут нужны пять конкретных операций (конструкций), позволяющих из данных предложений строить новые предложения. Исходя из элементарных формул (атомов), мы можем использовать эти операции снова и снова для построения других предложений – сложных формул, или молекул. Под формулами вообще мы будем понимать элементарные и сложные формулы. Если a и b – какие-либо данные формулы (т.е. либо атомы, либо уже построенные сложные формулы), то $a = b$, $a \supset b$, $a \wedge b$, $a \vee b$ также являются (сложными) формулами. Если a – данная формула, то $\neg a$ – также (сложная) формула (см. подразд. 1.6).

Рассмотрим структуру языка с точки зрения исчисления предикатов.

Описание знаний возможно осуществить с помощью совокупности дискретных объектов и отношений между ними. Объекты, если их соотнести с решаемой задачей, образуют ее предметную область. Например, если задача состоит в описании родственных отношений, то предметная область – множество людей, а если задача вычислительная, то предметной областью будет множество целых чисел.

Объекты при описании их средствами математической логики должны иметь имена. За определением имен следует описание соотношений между объектами и выражение свойств этих отношений. Построение решения задачи производится на основе логического вывода, манипуляцией предложениями, описывающими задачу.

Объекты, используемые в описании задачи, могут быть произвольными, например числовые или литеральные константы, переменные, принимающие значения из некоторого множества, функции.

Примеры констант: 1, 2, 33, 14565 – числовые константы; ab, АНТ25, ДОМ – литеральные константы.

Примеры переменных: *x*, *y*, *z*, X, ы, Д.

Примеры функций: автор(Толстой, Казаки), сумма (2,3).

В математической логике константы, переменные и функции объединены общим названием – терм. Понятие терма можно сформулировать следующим образом. *Терм* – это переменные, константы и функции вида

$$f(t_1,...,t_n),$$

где каждое t_i – терм, а f – *n*-арный функциональный символ или функтор (арность – это число аргументов).

Особенность функции состоит в том, что она принимает значение элемента предметной области D_n или, иными словами, представляет собой некоторое отображение совокупности из n (n-ки) элементов из предметной области в элемент предметной области.

Отношения, определяемые над объектами, отличаются от функций. Отношение определяет совокупность элементов из предметной области и представляет собой отображение из D_n в множество {ИСТИНА, ЛОЖЬ}. Например, отношение мама(*x*, *y*) определяет совокупность пар (*x*, *y*) таких, что элементы множества людей *x* и *y* находятся в отношении родства мама. Множество пар (*x*, *y*) – это множество матерей и детей. В математической логике отношениям даются имена, называемые предикатными символами, а сами отношения называются предикатами. В последнем примере предикатный символ – мама.

Более строго понятие предиката можно сформулировать следующим образом.

Предикат – это выражение вида $P(t_1,...,t_n)$, где каждое t_i – терм, а P – m-арный предикатный символ. Формально предикат $P(t_1,...,t_n)$ можно читать либо как "m-ка $(t_1,...,t_n)$ принадлежит отношению *P*", либо как "высказывание *P* справедливо для *m*-ки $(t_1,...,t_n)$".

В логике имеется набор связок: $\wedge$, $\vee$, $\neg$, $\leftarrow$, $\leftrightarrow$, которые читаются соответственно как "и", "или", "не", "если", "тогда и только тогда". Связки можно применить для образования логических формул, объединяя предикаты и другие формулы.

Наряду со связками существуют кванторы общности ($\forall$) и существования ($\exists$). Кванторы определяют пределы изменения переменных. Формула, стоящая после квантора, называется областью действия этого квантора.

Таким образом, понятие формулы можно сформулировать таким образом: *формула* – это либо предикат, либо выражение, составленное из формул с помощью логических связок и кванторов.

С помощью этого понятия можно сформулировать понятие предложения.

Предложение – это формула, в которой каждая переменная находится в области действия квантора общности.

В целях лаконичности записи предложений внешние кванторы общности будут опускаться.

В качестве примера предложения можно привести выражение: бабушка(x, y) $\leftarrow$ мама(x, z) $\wedge$ мама(z, y). Содержательно это предложение означает, что бабушка – это мама мамы. Перед этим предложением стоят знаки кванторов общности по переменным x, y, z. Их присутствие означает, что формула выполнена для всех x, y, z.

Предложения, построенные в соответствии с введенными выше правилами, образуют язык логики первого порядка. В этом языке термы представляют собой объекты, а предикаты – отношения между объектами. С помощью этого языка можно описать все задачи, решаемые на ЭВМ. На основе языка логики первого порядка можно построить различные языки логического программирования, различающиеся по правилам формирования предложения.

Используя принятую в данной области терминологию, введем несколько строгих определений.

Определение 1.5. **Грамматикой** называются правила, определяющие предложения языка.

Определение 1.6. **Фразами** называются комбинации символов, образующие грамматические единицы.

Определение 1.7. **Функтором** называется средство соединения фраз для образования других фраз.

Существуют три основных класса фраз: имена, предложения и функторы. Имена и предложения называют замкнутыми фразами, чтобы отличить их от функторов. Фразы, соединяемые функтором, называют его аргументами, а результат соединения – его значением.

Определение 1.8. **Конструкционная последовательность** – это последовательность, в которой каждый ее член либо является начальным элементом, либо строится из некоторых впереди стоящих элементов посредством некоторой операции.

Нормальная конструкционная последовательность получается из размеченной древовидной диаграммы нумерацией узлов сверху вниз и слева направо.

1.5. Грамматики

Традиционная лингвистика считала очень важным различие между "частными грамматиками", в компетенцию которых входило изучение специфических особенностей конкретных языков, и "универсальной грамматикой", занимавшейся общими свойствами человеческого языка как такового. С относительно недавнего времени указанное различие вновь стало привлекать внимание лингвистов. При этом оживление интереса к универсальной грамматике совпало с развертыванием серьезных исследований в той области математической лингвистики, которую иногда называют алгебраической лингвистикой – в отличие от статистических исследований языка или вероятностного моделирования речевого поведения.

До известного времени при изучении рекурсивных механизмов в синтаксисе естественных языков было просто не на что опереться: соответствующие формализмы еще не были созданы, а существовавший тогда аппарат не позволял достаточно четко и содержательно описывать эти механизмы. Основные необходимые понятия возникли и стали уточняться в 30-40-х годах в ходе исследований по основаниям математики. Более глубокое понимание природы рекурсивных механизмов и алгоритмов вообще, достигнутое за последние десятилетия, позволило вернуться к рассмотрению творческого аспекта речевой деятельности и предпринять попытку точно описать механизмы, которыми располагает каждый язык и которые необходимы для его свободного и непринужденного использования. В настоящее время дисциплина, занимающаяся изучением подобных механизмов, известна под названием теории порождающих грамматик. Строго говоря, здесь должна была бы идти речь не о порождающих грамматиках, а о формальных грамматиках, представляющих собой более широкий класс объектов: порождающие грамматики – это частный случай формальных грамматик (хотя и наиболее важный); существуют формальные грамматики иных типов, например распознающие. Определение порождающей грамматики, которое дает Н. Хомский, приложимо в

действительности к любой формальной грамматике. Порождающая грамматика того или иного естественного языка – это система правил, задающая потенциально бесконечное множество предложений данного языка и одновременно сопоставляющая каждому предложению описание его структуры, отражающее его существенные фонетические, синтаксические и семантические свойства.

Порождающая грамматика – это формальная грамматика, позволяющая построить любую правильную цепочку символов.

Построение порождающих грамматик стало возможным благодаря развитию математики; поэтому нет ничего удивительного в том, что интерес к формальным свойствам грамматик и к алгебраической лингвистике в целом пробудился именно в связи с этим новым подходом к исследованию языка.

Модели порождающих грамматик как части теории формальных грамматик были заложены в фундаментальных работах Хомского. Они были введены им как средство описания формальных языков (точнее, как средство описания формальных моделей естественных языков). Хомским определена так называемая "иерархия" классов грамматик и языков по степени сложности – грамматики и языки типа 0, 1, 2 и 3.

Определение 1.9. Грамматикой общего вида (типа 0) называется набор $G = (N,T,P,S)$, где N – алфавит нетерминальных символов (будем обозначать их большими латинскими буквами); T – алфавит терминальных символов, $T \cap N = \varnothing$. (терминальные символы будем обозначать малыми буквами из начала латинского алфавита); P – конечное множество правил (или продукций); правило имеет вид $\alpha \to \beta$, где $\alpha,\beta \in (T \cup N)^*$, причем α содержит хоть один нетерминальный символ; S – начальный нетерминальный символ, $S \in N$.

Символы объединенного алфавита $V = T \cup N$ будем обозначать малыми буквами из конца латинского алфавита (u, v, w, ...).

Основным понятием грамматик является понятие вывода и выводимости. Если в грамматике есть правило $\alpha \to \beta$, то из цепочки $\gamma\alpha\delta$ за один шаг выводима цепочка $\gamma\beta\delta$ (говорят также, что непосредственно выводима). Это отношение обозначается $\gamma\alpha\delta \to_G \gamma\beta\delta$. Отношение выводимости за 0 или более шагов

обозначается $\alpha \to_G^* \beta$ (рефлексивно-транзитивное замыкание отношения $\to_G$) и называется просто отношением выводимости. Последовательность цепочек $\alpha_0 \to_G \alpha_1 \to_G \ldots \to_G \alpha_n$ называется выводом длины $n \geq 0$. Вывод также будем обозначать иногда $\alpha \to_G^* \beta$. В случае, когда понятно, в какой грамматике рассматривается вывод, индекс *G* можно опускать.

Языком, порождаемым грамматикой *G*, называется множество терминальных цепочек, выводимых из *S*, т.е. множество $L(G) = \{\alpha \mid \alpha \in T, S \to^* \alpha\}$. Класс языков, порождаемых грамматиками общего вида, называется классом языков типа 0.

Грамматики *G* и *G'* называются эквивалентными, если они порождают один и тот же язык, т.е. $L(G) = L(G')$.

1.6. Исчисление высказываний: интуитивный подход

Теперь мы рассмотрим тот раздел логики, которая занимается изучением связей между высказываниями – связей, определяемых исключительно тем, каким образом одни высказывания строятся из других, элементарных, играющих при этом роль строительных блоков. Эту часть логики называют логикой высказываний или исчислением высказываний.

Высказывания. "Луна – спутник Земли", "Два плюс два – пять" – эти две фразы являются высказываниями. Первое высказывание истинно, его логическое значение – истина; второе высказывание ложно, его логическое значение – ложь.

"В этот день шел дождь", "Целое число n является простым" и т. п. – не высказывания, а пропозициональные ("высказывательные") переменные, которые в зависимости от обстоятельств могут принимать значения "истина" или "ложь".

Обычно значение "истина" обозначается единицей, а значение "ложь" – нулем (некоторые авторы используют обратные обозначения).

Таким образом, пропозициональные переменные суть булевы переменные; к этому факту мы еще вернемся.

Исчисление высказываний не занимается анализом высказываний, а рассматривает их как неразложимые единицы – с точки зрения возможностей по-разному комбинировать их с помощью различных логических операций, к обзору которых мы и перейдем.

Высказывания и пропозициональные переменные будем обозначать малыми латинскими буквами p, q, ...

Конъюнкция (логическое "и"). Высказывание "Идет дождь и дует ветер" считается истинным тогда и только тогда, когда одновременно идет дождь и дует ветер. Это высказывание является конъюнкцией высказываний "Идет дождь" и "Дует ветер".

Более точно, конъюнкция двух высказываний (или пропозициональных переменных) p и q есть высказывание (пропозициональная переменная), обозначаемое $p \wedge q$ и имеющее в четырех возможных случаях распределения значений p и q следующие логические значения:

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

Помимо символа $\wedge$ конъюнкцию обозначают также символами & и · или простым соположением высказываний.

Дизъюнкция (логическое "или"). Аналогичным образом вводится понятие дизъюнкции. Дизъюнкция определяется следующей таблицей:

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

Дизъюнкция $p \vee q$ считается истинной тогда и только тогда, когда истинно хотя бы одно из высказываний p или q. Дизъюнкция соответствует неразделительному употреблению русского "или" в такой фразе, как "Он дурак или мерзавец"; подразумевается "А может быть и то, и другое".

Дизъюнкцию обозначают иногда также символом +.

Отрицание. Любому высказыванию (пропозициональной переменной) p можно поставить в соответствие высказывание $\neg p$ – его отрицание, которое определяется следующей таблицей:

p	$\neg p$
0	1
1	0

Другие обозначения отрицания: $\overline{p}$, $\sim p$.

Импликация ("влечет", "если..., то..."). Любой паре высказываний p, q можно поставить в соответствие высказывание $p \supset q$, определяемое следующей таблицей:

p	q	$p \supset q$
0	0	1
0	1	1
1	0	0
1	1	1

Напоминаем ещё раз, что высказывания p и q не анализируются, их содержание не берется в расчет ни в какой степени, и единственное, что о них известно, – это их логические значения. Поэтому импликация, которая была только что определена, не имеет никакого отношения к причинно-следственной зависимости или к выводу q из p (к доказательству q на основе p). Логический оператор $\supset$ в выражении "$p \supset q$" ни в коем случае нельзя смешивать с метаимпликацией "$p \Rightarrow q$" (означающей, что q действительно выводимо из p).

Пример 1.2.

p : Наполеон умер на острове Св. Елены.

q : Верцингеторикс носил усы.

$p \supset q$ принимает значение q.

Пример 1.3.

p : $2 \times 2 = 5$.

q : 12 – простое число.

$p \supset q$ принимает значение 1.

Пример 1.4.

p : Луна состоит из овечьего сыра.

q : 17 – простое число.

$p \supset q$ принимает значение 1.

Пример 1.5.

p : 17 – простое число.

q : 16 – простое число.

$p \supset q$ принимает значение 0.

Правило вывода (modus ponens) гласит: если мы знаем (откуда – нас сейчас не касается!), что p имеет значение 1 и $p \supset q$ имеет значение 1, то мы можем сделать вывод, что и q имеет значение 1.

Логическая эквивалентность., Из двух высказываний (или пропозициональных переменных) p и q можно образовать высказывание $p \equiv q$, определяемое таблицей:

p	q	$p \equiv q$
0	0	1
0	1	0
1	0	0
1	1	1

Здесь фактически представлено высказывание "p эквивалентно q" или, что то же самое, "p имеет то же логическое значение, что и q".

Как и в случае импликации, при образовании логической эквивалентности реальное содержание соединяемых высказываний совершенно не принимается во внимание.

Классификация логических выражений. Пусть у нас имеется конечное или бесконечное число пропозициональных переменных p, q, ... ; будем образовывать из них выражения последовательным применением логических операций $\wedge$, $\vee$, $\neg$, $\supset$, $\equiv$.

Пример 1.6.

$$(p \vee q) \wedge q, \quad (p \supset q) \vee (q \supset p)$$

Существуют выражения, которые принимают значение 1, каковы бы ни были значения входящих в них переменных.

Пример 1.7.

$p \vee \neg p$, $p \supset p$

В силу якобы парадоксального характера нашей импликации таким является и выражение

$$p \supset (q \supset p)...$$

Такие выражения называются тождественно истинными, или тавтологиями.

Существуют выражения, которые принимают значение 0, каковы бы ни были значения входящих в них переменных.

Пример 1.8.

$$p \wedge \neg p,\ (p \vee \neg p) \supset (p \wedge \neg p)$$

Такие выражения называются тождественно ложными, или противоречивыми.

Существуют, наконец, выражения, которые принимают то значение 1, то значение 0 в зависимости от того, каковы значения входящих в них переменных.

Например, $(p \supset q) \vee (p \wedge q)$. В самом деле, обозначив это высказывание через r, мы получим

p	q	r
0	0	1
0	1	1
1	0	0
1	1	1

Поиск тавтологий представляет особый интерес, поскольку в логике тавтологии играют ту же роль, что в алгебре – тождества.

Пример 1.9. Выражение

$$(\neg q \supset \neg p) \supset (p \supset q)$$

является тавтологией (что нетрудно доказать, построив таблицу истинности). Эта тавтология лежит в основе правила контрапозиции.

Тавтологическая эквивалентность. Мы будем писать $r = s$, желая выразить тот факт, что высказывание $r \equiv s$ является тавтологией.

Пример 1.10.

$$(p \vee q) \vee m = p \vee (q \vee m);$$
$$(p \wedge q) \wedge m = p \wedge (q \wedge m);$$
$$p \supset q = \neg p \vee q;$$
$$p \equiv q = (p \supset q) \wedge (q \supset p).$$

Отношение = (называемое тавтологической эквивалентностью) означает, что при любых условиях выражения, стоящие слева и справа от символа =, принимают одно и то же логическое значение. Это отношение является отношением эквивалентности (иначе говоря, оно рефлексивно, симметрично и транзитивно).

Константа "истина" тавтологически эквивалентна любой тавтологии. Например:

$$p \vee \neg p \equiv \text{истина}.$$

Константа "ложь" тавтологически эквивалентна любому противоречивому выражению. Например:

$$p \wedge \neg p \equiv \text{ложь}.$$

Правила де Моргана. Легко убедиться в том, что выражения

$$\neg(p \vee q) = \neg p \wedge \neg q; \qquad \neg(p \wedge q) = \neg p \vee \neg q$$

являются тавтологическими эквивалентностями. Эти эквивалентности называются правилами де Моргана.

Для любых переменных или выражений r, s, t имеем:

1) $r \vee r = r$; $r \wedge r = r$: идемпотентность;

2) $r \vee s = s \vee r$; $r \wedge s = s \wedge r$: коммутативность;

3) $r \vee (s \vee t) = (r \vee s) \vee t$; $r \wedge (s \wedge t) = (r \wedge s) \wedge t$: ассоциативность;

4) $r \vee (r \wedge s) = r = r \wedge (r \vee s)$;

5) $\left.\begin{array}{l} r \vee (s \wedge t) = (r \vee s) \wedge (r \vee t) \\ r \wedge (s \vee t) = (r \wedge s) \vee (r \wedge t) \end{array}\right\}$: дистрибутивность;

6) $r \vee \neg r = \text{истина}$; $r \wedge \neg r = \text{ложь}$;

7) $\left.\begin{array}{l}\neg(r \vee s) = \neg r \wedge \neg s \\ \neg(r \wedge s) = \neg r \vee \neg s\end{array}\right\}$: двойственность (правила де Моргана);

8) $\neg(\neg r) = r$: инволютивность отрицания.

Соотношения 1 – 7 показывают, что исчисление высказываний снабжает множество логических выражений структурой булевой алгебры – это следует из самого определения булевой алгебры. Точнее, мы имеем здесь в виду выражения, полученные из элементов счетного множества переменных p, q ... путем применения унарной операции $\neg$ и бинарных операций $\vee$ и $\wedge$ произвольное число раз, причем тавтологически эквивалентные выражения считаются равными.

Теоретико-множественная интерпретация исчисления высказываний. Возьмем семейство множеств P, Q, ... и будем интерпретировать пропозициональные переменные p, q, ... как выражения вида $x \in P$, $x \in Q$, ... с одним и тем же x.

Тогда $p \wedge q$ интерпретируется как $x \in P \cap Q$, $p \vee q$ – как $x \in P \cup Q$ и т. п.

Исчисление высказываний получает теперь следующую интерпретацию. Его теоретико-множественным образом является алгебра множеств, получаемых из P, Q, ... с помощью операций пересечения, объединения и взятия дополнения относительно некоторого универсального множества U, содержащего все P, Q, ... и соответствующего истине. Взятие дополнения относительно U соответствует отрицанию; пустое множество $\varnothing$, являющееся дополнением для U, соответствует лжи.

Варианты исчисления высказываний. Можно строить исчисления, эквивалентные описанному выше, но с другими основными операциями.

Один из наиболее интересных вариантов исчисления высказываний получается в случае, когда используются только операции $\neg$ и $\supset$. Мы знаем, что $p \supset q = \neg p \vee q$ и что $\neg\neg p = p$, откуда $p \vee q = \neg p \supset q$.

Выражение для $p \wedge q$ можно получить с помощью правила де Моргана.

Подчеркнем, что все операции исчисления высказываний могут быть выражены через одну подходящим образом выбранную операцию.

1.7. Исчисление высказываний: формальный подход

Алфавит. Пусть имеется счетное множество абстрактных математических объектов (базовых элементов), образующих абстрактный алфавит. Эти объекты находятся во взаимно однозначном соответствии со следующими графическими символами, образующими алфавит:

$$A = \{], [, \neg, \supset, a_1, ..., a_n, ...\}.$$

Здесь "{", "}" и "," – символы метаязыка; "]" и "[" – имена формальных скобок; " $\neg$ " – имя формального отрицания; " $\supset$ " – имя формальной импликации; a_i – имена элементарных объектов, называемых атомами и образующих счетное множество.

Множество, снабженное всюду определенной ассоциативной операцией, в алгебре называется полугруппой. Снабдив множество A^* операцией конкатенации (соединения цепочек), мы превращаем его в свободную полугруппу над A, причем элементы A являются образующими этой полугруппы.

Пример 1.11. Если $A = \{0,1\}$, то цепочки множества A^* можно трактовать как целые двоичные числа; тогда конкатенацию можно интерпретировать как некоторую сложную арифметическую операцию.

Формулы. Мы выделим в полугруппе A^* некоторую часть – формальный язык, соответствующий некоторому множеству конкретных формул (служащих представлениями абстрактных формул).

Правила, задающие это множество формул, предписывают определенное "правильное" употребление скобок и символов операций. Вот эти правила:

*F*1. Всякий атом есть формула.

*F*2. Если слово X – формула, то слово $\neg X$ – тоже формула.

*F*3. Если слова X и Y – формулы, то слово $[X \supset Y]$ – тоже формула.

*F*4. Никаких других формул нет.

X и Y являются здесь метаязыковыми переменными.

Пример 1.12.

a_1 является формулой в соответствии с *F*1;

a_2 является формулой в соответствии с *F*1;

$[a_1 \supset a_2]$ является формулой в соответствии с *F*3;

$\neg[a_1 \supset a_2]$ является формулой в соответствии с *F*2;

a_3 является формулой в соответствии с *F*1;

$[\neg[a_1 \supset a_2] \supset a_3]$ является формулой в соответствии с *F*3 и предыдущим замечанием.

Множество всех формул можно получить, последовательно строя формулы все большей длины.

Формулы длины 1 – это атомы. Поскольку в формулах ничего не стирается, каждая формула содержит по крайней мере один атом.

Формулы длины 2 таковы:

$$\neg a_1, \neg a_2, \ldots, \neg a_n, \ldots$$

затем идут формулы длины 3:

$$\neg\neg a_1, \neg\neg a_2, \ldots, \neg\neg a_n, \ldots$$

формулы длины 4:

$$\neg\neg\neg a_1, \neg\neg\neg a_2, \ldots, \neg\neg\neg a_n, \ldots$$

формулы длины 5:

$$\neg\neg\neg\neg a_1, \neg\neg\neg\neg a_2, \ldots, \neg\neg\neg\neg a_n, \ldots$$

и

$$[a_1 \supset a_1], [a_1 \supset a_2], \ldots, [a_n \supset a_p], \ldots$$

и т. д.

Если дано слово $M \in A^*$, то всегда можно решить, является ли M формулой.

Если M не содержит атомов, то M не является формулой. Если M содержит атомы, то нужно построить из этих атомов все возможные

формулы длины, меньшей или равной длине M. Слово M является формулой тогда и только тогда, когда M является одной из этих формул. Ясно, что указанный процесс определения того, является ли M формулой, может быть "механизирован". При этом для нас здесь неважно, насколько такой процесс "удобен" или "эффективен"; этот вопрос обсуждаться не будет.

Пример 1.13.

$\neg[[$ не является формулой (в данном выражении нет атомов);

$[\neg a_2 \supset a_2]$ является формулой, поскольку это выражение содержится в множестве всех формул длины ≤ 6, которые можно построить из a_2; это множество состоит из следующих формул:

$$a_2$$
$$\neg a_2$$
$$\neg\neg a_2$$
$$\neg\neg\neg\neg a_2, [a_2 \supset a_2]$$
$$\neg\neg\neg\neg\neg a_2, \neg[a_2 \supset a_2], [\neg a_2 \supset a_2], [a_2 \supset \neg a_2]$$

Множество элементарных теорем. Теперь мы рассмотрим другой (формальный) язык, содержащийся в предыдущем; этот язык мы назовем множеством теорем. Теоремы соответствуют тому, что в неформальном варианте называлось тавтологиями.

Сначала мы определим элементарные теоремы с помощью следующих соглашений.

Если слова X, Y, Z – формулы, то:

*Т*1. $[X \supset [Y \supset X]]$ есть элементарная теорема.

*Т*2. $[[\neg X \supset \neg Y] \supset [Y \supset X]]$ есть элементарная теорема.

*Т*3. $[[X \supset [Y \supset Z]] \supset [[X \supset Y] \supset [X \supset Z]]]$ есть элементарная теорема.

*Т*4. Других элементарных теорем нет.

Если рассматривать элементарные теоремы как аксиомы, то выражения *Т*1–*Т*3, принадлежащие метаязыку, будут схемами аксиом.

Легко убедиться в том, что элементарные теоремы являются формулами в силу *F*2 и *F*3. Нетрудно показать также, что для любого слова *M* можно решить, является это слово элементарной теоремой или нет.

О выборе элементарных теорем. Отметим, что *T*1 отражает "квазипарадоксальный" характер импликации. *T*2 есть правило контрапозиции. Необходимо особо обратить внимание на следующий факт: когда мы используем содержательное понятие истины, ясно, что

$$\neg(\neg p) = p,$$

поэтому в этом случае безразлично, писать

$$[\neg X \supset \neg Y] \supset [Y \supset X]$$

или

$$[Y \supset X] \supset [\neg X \supset \neg Y].$$

Однако дело обстоит совсем не так, когда мы с самого начала не прибегаем к понятию истины (т. е. когда мы имеем в виду другую интерпретацию).

Выбор *T*3 обусловлен техническим удобством.

Исходя из элементарных теорем, мы будем строить теоремы, задав a priori правило вывода.

Правило вывода. Если слова X и $[X \supset Y]$ суть теоремы, то слово Y – тоже теорема.

Легко видеть, что теоремы обязательно являются формулами. Заметим, что введенное нами правило есть известное правило modus ponens.

Пример 1.14.

$[a_1 \supset a_1]$ и вообще $[X \supset X]$ суть теоремы.

Для доказательства построим теорему, в которой $[a_1 \supset a_1]$ окажется в позиции Y из правила вывода.

(1) a_1 является формулой в силу *F*1.

(2) Слово $[a_1 \supset a_1]$, которое мы будем обозначать впредь через M, является формулой в силу (1) и *F*3.

(3) Слово $[a_1 \supset M]$ является формулой в силу (1), (2) и *F*3.

(4) Слово $[a_1 \supset M]$ является теоремой в силу (1) и *T*1.

(5) Слово $[a_1 \supset [M \supset a_1]]$ является теоремой в силу (1), (2) и *T*1.

(6) Слово $[a_1 \supset [M \supset a_1]] \supset [[a_1 \supset M] \supset [a_1 \supset a_1]]$, структура которого может быть изображена деревом (рис. 1.2), является теоремой в силу (1), (2) и *T*3.

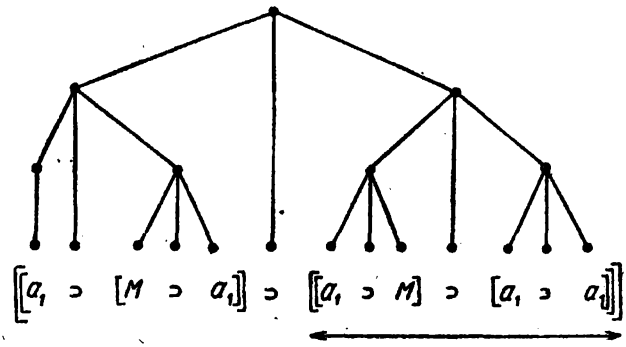


Рис. 1.2

(7) Правая часть слова (6), под которой на рис. 1.2 помещена стрелка, является теоремой в силу (5), (6) и правила modus ponens.

(8) $[a_1 \supset a_1]$ является теоремой в силу (4), (7) и правила modus ponens.

Точно так же можно доказать, что для любой формулы X выражение $[X \supset X]$ является теоремой.

Итак, соблюдая некоторые "правила игры", мы построили систему, состоящую из двух связанных частей. Мы можем придать этой системе определенный смысл: она формализует "исчисление высказываний, рассматриваемое на интуитивном уровне понимания"; стало быть, она формализует в какой-то степени "логику здравого смысла".

Заметим, что построенная система не зависит от интерпретации, которую мы ей даем; возможны и другие интерпретации.

Теперь остается показать, каким образом в рамках данной формальной системы можно было бы, действуя вполне строго, ввести истинностные значения; однако мы этого делать не будем.

1.8. Определение формальной системы

Определение 1.10. Мы будем называть формальной (логистической) системой упорядоченную четверку множеств

$$S = \langle X, L, A, R \rangle,$$

где X – счетный алфавит, буквы которого могут считаться перенумерованными; L –(формальный) язык над X, или множество формул ($L \subset X$); A – некоторое подмножество множества L, или множество аксиом, задаваемое с помощью схем аксиом; R –множество правил вывода (их вид уточняется ниже).

Правила вывода. Рассмотрим декартово произведение $L \times L \times ... \times L$, состоящее из n сомножителей ($n \geq 1$); его элементами являются упорядоченные n-ки формул.

Правило вывода R есть правило, позволяющее поставить в соответствие каждой n-ке $\langle y_1, y_2, ..., y_n \rangle$ из некоторого подмножества произведения $L \times L \times ... \times L$ одну определенную формулу x.

Более точно, правило R определяется некоторым подмножеством декартова произведения $L^n \times L$, причем это подмножество должно обладать следующим свойством: ни для какой n-ки $\langle y_1, y_2, ..., y_n \rangle \in L^n$ не может существовать более одной формулы $x \in L$, для которой $\langle \langle y_1, y_2, ..., y_n \rangle, x \rangle$ принадлежит данному подмножеству. Областью определения правила R является некоторое подмножество L^n.

При конкретном применении правила вывода формулы $y_1, y_2, ..., y_n$ называют посылками, а формулу x –заключением; посылки и заключение вместе суть аргументы примененного правила.

Выводимость. Пусть $\Phi = \langle f_1, f_2, ..., f_k \rangle$ – упорядоченное множество (последовательность) формул. Если существуют правило R и формула x такие, что $\Phi R x$, то x называется непосредственным следствием из Φ.

Множество правил вывода R определяет, таким образом, отношение непосредственной выводимости: ΦRx тогда и только тогда, когда существует правило вывода R такое, что ΦRx. Отношение R определяется в декартовом произведении множества последовательностей формул на множество формул.

Пусть M – произвольное множество формул. Обозначим через E_0 объединение M и множества аксиом. Образуем всевозможные конечные последовательности элементов E_0 и добавим к E_0 все формулы, являющиеся непосредственными следствиями из таких последовательностей. Полученное множество обозначим через E_1. Построим, далее, множество E_2 с помощью E_1, точно так же, как E_1 строилось с помощью E_0, и т. д. Формулы, принадлежащие $E_1, E_2, ..., E_n, ...$, будем называть выводимыми из M, или следствиями из M. Выделяя формулы, действительно участвующие в получении выводимой из M формулы y, получим вывод этой формулы из M. (Точнее, вывод есть дерево, строящееся следующим образом: корнем его служит y; в концах дуг, исходящих из корня, стоят формулы, непосредственным следствием которых является y, и т. д.; висячие вершины дерева будут элементами E_0.) Запись $M \models y$ означает, что "y выводима из M".

Замечания.

1. M определяется только составом своих элементов; их порядок значения не имеет. Множество M называется множеством посылок.

2. Если множество N содержит все элементы из M ($M \subset N$) и $M \models y$, то ясно, что $N \models y$; иначе говоря, к посылкам можно присоединять любые другие.

3. Из непосредственной выводимости следует выводимость.

4. Всякая формула выводима из самой себя.

Теоремы. Формулы, выводимые из аксиом некоторой формальной системы, называются ее теоремами.

Характеристическое свойство теорем состоит в том, что они выводимы из любого множества посылок, в том числе из пустого множества. Вместо $\varnothing \models y$ пишут сокращенно $\models y$.

Приведем примеры формальных систем.

Комбинаторные системы – это **формальные системы** особого типа. Как показывает их название, они используются при исследовании задач комбинаторного характера, а именно, относящихся к словам.

Если алфавит X формальной системы конечен, L совпадает с X и имеется ровно одна аксиома, то S представляет собой комбинаторную систему.

Комбинаторная система – это формальная система, в которой:

1) множество элементарных теорем содержит ровно одно слово – аксиому;

2) правила вывода задаются продукциями.

Другими примерами формальных систем являются:

N – формальная арифметика (см. подразд. 4.5);

G – формализация теории групп;

AG – абелевы группы;

Pp – чистое исчисление высказываний;

Pd – чистое исчисление предикатов;

Pd= – чистое исчисление предикатов с равенством;

Pdf – чистое исчисление предикатов с функциями;

Pdf= – чистое исчисление предикатов с функциями и равенством.

Кроме того, мы можем также рассматривать формальные исчисления высказываний и исчисления предикатов (с равенством и без), использующие правила образования каких-либо более сложных или более частных систем.

2. АЛГОРИТМЫ КАК ФОРМАЛЬНЫЕ СИСТЕМЫ

2.1. Интуитивное понятие алгоритма

Одним из главнейших понятий математики является, как известно, понятие функции. Содержательно функция представляет собой "закон соответствия", по которому каждому элементу области определения – значению аргумента – сопоставляется какой-то элемент области прибытия – значение функции. При этом, вообще говоря, не требуется, чтобы значение функции можно было каким-либо способом "в самом деле найти" ("построить", "вычислить" и т. п.) по соответствующему значению аргумента; однако ясно, что для математики и в особенности для ее приложений все же должны быть наиболее интересны те функций, для которых способы нахождения (иначе – "построения" или "вычисления") их значений существуют.

В трудах Ньютона и Лейбница, в которых впервые появилось понятие функции, равно как и в работах математиков следующих поколений, неявно предполагалось, что значение функции всегда можно так или иначе "найти", производя над значением аргумента некоторые "действия", предписываемые связанным с этой функцией "аналитическим выражением". Правда, тогда рассматривались только функции действительной и комплексной переменной; но потом, в XIX столетии, было замечено, что даже если ограничиться только такими функциями, их общие свойства никак не связаны с возможностью "вычислять" их значения (более того, с вычислимостью не связаны и те специальные свойства, которые играют основную роль в анализе – непрерывность, дифференцируемость и т. п.). Тогда понятие функции было освобождено от связи с аналитическим выражением и приобрело ту общность, которой мы пользуемся теперь. Но осознание того факта, что не для всех функций можно "вычислить" их значения, не могло, разумеется, уменьшить практического интереса к изучению тех функций, для которых такое вычисление возможно; с теоретической же точки зрения естественно было теперь поставить вопрос о том, чем отличаются "вычислимые" функции от "невычислимых", т. е. о выработке точного понятия способа вычисления функции.

Интуитивное представление о "вычислительной процедуре" существовало у математиков уже давно, и за этими процедурами был закреплен специальный термин, не отвечавший еще никакому точному понятию, – алгоритм.

Слово "алгоритм" происходит от имени работавшего в IX в. в Багдаде (и писавшего, как все тогдашние мусульманские ученые, по-арабски) математика аль-Хорезми (это имя указывает на происхождение из Хорезма). В латинских переводах его сочинений, получивших распространение в Европе с XII века, имя автора писалось в искаженной форме, чаще всего Algorithmus или Algorismus. По арифметическому трактату аль-Хорезми европейцы впервые познакомились с созданной в Индии позиционной системой счисления, которой мы пользуемся и сейчас. Со временем слово Algorithmus превратилось в название основанной на этой системе арифметики, изданной тогда в виде собрания правил действий над числами. Затем так стали называть любую систему правил, позволяющих чисто механически выполнить любую конкретную операцию некоторого определенного типа. Можно охарактеризовать алгоритм еще и как такую механическую процедуру, которая может применяться к символам некоторого класса (входным символам) и, возможно, выдает для данного входного символа определенный выходной символ.

18.

Интуитивное понятие алгоритма представляет его как строго сформулированное правило, задающее последовательность выполнения конечного числа шагов, следуя которому можно осуществить процесс решения задачи, причем осуществить механически, максимально экономя усилия.

Алгоритм (алгорифм) – способ (программа) решения вычислительных и других задач, точно предписывающий, как и в какой последовательности получить результат, однозначно определяемый исходными данными.

Алгоритм задается как предложение формального языка.

Алгоритмический язык – это формализованный язык для однозначной записи алгоритмов.

Пример 2.1. Дифференцирование многочлена.

(1) Взять очередной член многочлена; если такового уже не осталось, написать "Конец" и остановиться.

(2) Взять показатель степени переменной данного члена и умножить его на коэффициент, сделав это произведение коэффициентом при новом члене; уменьшить его на 1 и сделать показателем степени при новом члене.

(3) Вернуться к (1).

Пример 2.2. Поиск слова в двуязычном (переводном) словаре.

(1) Взять первую букву искомого слова и найти тот раздел словаря, первое слово которого начинается этой буквой.

(2) Взять первые три буквы искомого слова и найти в найденном на шаге (1) разделе первую из страниц, на которых имеются слова, начинающиеся этими буквами.

(3) Начиная с найденной страницы, просматривать подряд все слова, сравнивая их с искомым словом до тех пор, пока либо будет получено совпадение ("искомое слово найдено"), либо мы дойдем до слова, не начинающегося на три буквы, выделенные на шаге (2) ("искомое слово не найдено").

(4) Если искомое слово найдено, взять все его эквиваленты; конец.

(5) Если искомое слово не найдено, взять более полный словарь и вернуться к (1).

Несколько более сложные примеры – алгоритм Евклида для нахождения наибольшего общего делителя двух натуральных чисел, правило Крамера для решения систем линейных уравнений, правила дифференцирования элементарных функций и т. п.

Кроме алгоритмов для вычисления функции, говорят еще об алгоритмах для распознавания свойств и отношений, подразумевая под этим строго определенные "правила" или "процедуры", позволяющие

отвечать на вопросы вроде "Является ли данное натуральное число простым?", "Являются ли два данных натуральных числа взаимно простыми?" и т. п. Однако задача распознавания свойства или отношения всегда может быть представлена как задача нахождения, или "вычисления", истинностного значения некоторого предиката (одноместного и многоместного); поэтому под алгоритмом всегда можно понимать процедуру, позволяющую "находить" или "вычислять" значения некоторой функции.

Процедуры, описанные в обоих примерах, являются механическими (в интуитивном смысле этого слова). Однако если бы мы захотели реализовать эти процедуры с помощью машины, лишенной сообразительности, знаний о мире и т. д., нам пришлось бы заняться их уточнением, поскольку многие инструкции в данном описании не сформулированы в явном виде.

Анализ только что приведенных и любых других примеров алгоритмов показывает, что все они обладают следующими двумя характерными особенностями:

• наличие четкого описания. Алгоритм всегда описывается настолько четко, что на основе этого описания можно действовать чисто механически, не вникая в смысл;

• детерминированность. Это значит, что если процедура является алгоритмом, то ни в какой момент ее выполнения не должно возникать сомнения, что делать дальше: каждый следующий шаг должен однозначно определяться исходными данными (т. е. значением аргумента) и результатами уже выполненных шагов.

Более детальная характеристика понятия "алгоритм" должна включать следующие аспекты (по Хартли Роджерсу):

1) алгоритм есть совокупность инструкций, имеющих конечную длину (эта длина измеряется, например, числом символов, необходимых для записи правил);

2) имеется механизм (человек, механическое, оптическое и т. п. устройство, электронная машина), воспринимающий и выполняющий инструкции;

3) имеются средства (человеческая память, карандаш и бумага, зубчатые колеса, магнитные запоминающие устройства и т. п.), позволяющие фиксировать и хранить сведения о любых этапах работы по выполнению инструкций, а также выдавать эти сведения по мере необходимости;

4) существенно, что все выполняемые процедуры являются дискретными (хотя обрабатываемые ими объекты могут быть по своему характеру непрерывными);

5) последовательность элементарных операций, из которых складываются инструкции, жестко определена (как обычно говорят, детерминирована): на каждом шаге процесса переход к следующему шагу возможен не более чем одним способом.

Здесь прослеживается аналогия с парой (вычислительная машина, программа), а именно:

а) соответствует программе;

б) соответствует вычислительной машине, работающей по этой программе;

в) соответствует машинной памяти;

г) соответствует дискретной ("цифровой") природе цифровых вычислительных машин (в отличие от аналоговых);

д) соответствует механичности и жесткой детерминированности порядка выполнения команд программы.

Неформально и схематически алгоритм представлен на рис. 2.1.

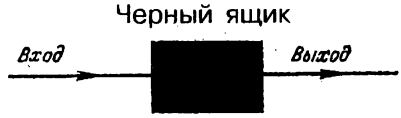


Рис. 2.1

На вход подается информация или объект, подлежащий обработке, например многочлен в примере 2.1, а выходом является результат обработки или операции над входом (производная многочлена в том же примере). Выход вырабатывается автоматически черным ящиком, который может быть компьютером или школьником, действующим по инструкции, или даже очень умной, хорошо тренированной собакой. Алгоритм есть процедура (или способ вычисления), осуществляемая черным ящиком для получения выхода из входа.

Сказанное сейчас ни в коей мере не может еще претендовать на статус математического определения; мы хотели только подытожить, хотя бы частично, давно существующие в математике интуитивные представления об алгоритме, или "вычислительной процедуре". Этими интуитивными представлениями математики могли обходиться до тех пор, пока речь шла только об отыскании алгоритмов для решения конкретных "вычислительных" задач и при этом всякий раз нужный алгоритм удавалось найти – потому что когда алгоритм найден, ни у кого из математиков не возникает сомнения, что это действительно алгоритм, т. е. вычислительная процедура. Но со временем возникли некоторые задачи несомненно "вычислительного" характера, все

попытки нахождения алгоритмов для которых заводили в тупик, и появилась гипотеза, что по крайней мере для части этих задач вообще никаких алгоритмов не существует. Проверка этой гипотезы в каждом случае требует уже строгого определения алгоритма: ведь если мы хотим математически доказать, что чего-то не существует, это "что-то" должно быть определено так, как полагается определять математические понятия.

Именно в связи с проблемой доказательства несуществования алгоритма в 30-е годы нынешнего столетия одновременно несколькими исследователями – К. Гёделем, Э. Постом, А. Тьюрингом – были предприняты усилия дать точное определение соответствующего понятия.

В результате этих усилий было предложено несколько вариантов определения, которые оказались равносильными, и на их основе удалось доказать, что для многих "вычислительных" задач алгоритмы действительно невозможны. Вместе с тем благодаря наличию строгого определения математики смогли заняться изучением общих свойств алгоритмов (или, что по существу то же самое, "вычислимых" функций), их классификацией и т.п.

Теория алгоритмов возникла в математической логике как вспомогательная дисциплина, предназначенная для исследования проблемы неразрешимости и обоснования математики.

2.2. Формализация и обобщение понятия алгоритма

В настоящее время существует много вариантов определения понятия алгоритма. Все они представляют собой довольно сложные, развернутые конструкции и по форме нередко различаются весьма существенно. Поэтому они обычно трактуются как самостоятельные математические понятия, и каждое из них имеет свое название; доказательство равносильности двух таких конструкций (а все они между собой равносильны) иногда оказывается не слишком простым. Две из этих конструкций получили особенно широкое распространение ввиду их наглядности и удобства.

Одна из них, названная в честь ее автора машиной Тьюринга, будет рассмотрена в подразд. 3.4, другая – в подразд. 4.3 и 4.4. Из остальных конструкций наиболее распространенной (и также очень естественной и наглядной) является нормальный алгоритм А. А. Маркова, который будет рассмотрен в подразд. 2.3.

Отмеченное в подразд. 2.1 условие наличия четкого описания алгоритма предполагает, очевидно, что все объекты, с которыми

алгоритм имеет дело – значения аргумента вычисляемой функции, значения самой функции, возникающие в процессе вычисления "промежуточные данные", – также должны быть описаны настолько четко, чтобы с ними можно было оперировать механически.

Но любые "четко описываемые" математические объекты допускают представление в виде слов в алфавитах; поэтому можно потребовать, чтобы именно слова в алфавитах были теми объектами, с которыми работает алгоритм. Для машин Тьюринга дело обстоит именно так. Имеются и другие универсальные способы представления "четко описанных" объектов; в частности, их можно представлять с помощью натуральных чисел, и некоторые варианты понятия алгоритма работают не со словами, а с натуральными числами, например рассматриваемые ниже, в подразд. 4.3 и 4.4, рекурсивные функции.

2.3. Марковские алгоритмы

В данном подразделе дается представление об одном подходе к уточнению понятия алгоритма, предложенном А.А. Марковым и называемом "нормальные алгоритмы". Данный подход связывает неформальное понятие эффективности с переработкой слов в некотором конечном алфавите в соответствии с определенными правилами. В качестве элементарного преобразования используется подстановка одного слова вместо другого. Множество таких подстановок определяет схему алгоритма. Правила, определяющие порядок применения подстановок, а также правила останова являются общими для всех нормальных алгоритмов. Дадим формальное определение.

Определение 2.1. Пусть $A = \{a_1, ...a_n\}$ – алфавит. Если P, Q – слова в алфавите A (возможно, пустые), то выражения $P \to Q$, $P \to (\cdot)Q$ называются формулами подстановки в алфавите A (предполагается, что знаки $\to$, $\cdot$ не входят в слова в алфавите A). При этом формула $P \to Q$ называется простой, а формула $P \to (\cdot)Q$ – заключительной. Обозначим через $P \to (\cdot)Q$ любую из этих формул. Произвольная конечная последовательность таких формул называется схемой S_a нормального алгоритма a. Таким образом, схема нормального алгоритма имеет вид:

$$P_1 \to (\cdot)Q_1$$
$$P_2 \to (\cdot)Q_2$$
$$\ldots$$
$$P_m \to (\cdot)Q_m$$

Схема S_a определяет следующий алгоритм a, перерабатывающий слова в алфавите A (т.е. вычисляющий словарную функцию на словах в алфавите A). Говорят, что слово P входит в слово W, если существуют такие слова V_1 и V_2 (возможно, пустые), что

$$W = V_1 P V_2 . \qquad (*)$$

Если слово V_1 имеет наименьшую длину из всех слов вида (*), то говорят о первом вхождении P в слово W.

Пусть дано произвольное слово K в алфавите A. Возможны следующие случаи:

1) ни одно из слов $P_1,\ldots,P_m$ не входит в слово K. В этом случае говорят, что схема S_a неприменима к K и пишут $S_a : K$;

2) среди слов $P_1,\ldots,P_m$ существует P_i, входящее в K. Пусть t – минимальное число, такое, что P_t входит в K, и пусть $K = V_1 P_t V_2$, где V_1 имеет минимальную длину (т.е. берется первое вхождение P_t в K). Тогда определим слово $W = V_1 Q_t V_2$. В этому случае говорят, что схема S_a применима к K и пишут

$$S_a : K \Rightarrow W;$$
$$S_a : K \Rightarrow \cdot W$$

для простой и заключительной формулы соответственно. Теперь пишем $S_a : K \Rightarrow W$, если существует конечная последовательность слов $W_0, W_1, \ldots, W_k$ в алфавите A такая, что $K = W_0$, $W = W_k$ и выполнено условие $W_0 \Rightarrow W_1$, $W_2 \Rightarrow W_3, \ldots, W_{k-1} \Rightarrow \cdot W_k$ либо $W_{k-1} \Rightarrow W_k$.

В первом случае пишут также $S_a : K \Rightarrow W$.

Нормальный алгоритм (по Маркову) – это ряд предписаний вида $A_i \to B_i$ ($i = 1,2,...,n$), где $A_1,...,A_n$, $B_1,...,B_n$ – фиксированные выражения в определенном алфавите, а каждая строчка (предписание) называется командой алгоритма.

Нормальный алгоритм a со схемой S_a вычисляет словарную функцию $F_s : A^* \to A^*$, где A^* – множество слов в алфавите A, если для любых слов $P, Q \in A^*$ имеем

$$F_S(P) = Q \Leftrightarrow \begin{cases} \text{либо } S_a : P \Rightarrow Q \text{ и } S_a : Q \\ \text{либо } S_a : P \Rightarrow Q \end{cases}.$$

Работа нормального алгоритма может быть описана следующим образом. Если дано слово P, то находим в схеме алгоритма S_a первую формулу $P \to (\cdot)Q_t$ такую, что P_t входит в P, и производим замену первого вхождения P_t словом Q_t. Пусть P_t – результат такой подстановки. Если применяемая формула $P \to \cdot Q_t$ – заключительная, то работа алгоритма заканчивается и слово W_1 есть результат работы алгоритма. Если применяемая формула $P \to Q_t$ – простая, то описанная процедура применяется к слову W_1. Если на некотором шаге получено слово W_i, к которому неприменима схема алгоритма S_a (т.е. ни одно из P_j, $j \in 1,...,m$ не входит в W_i), то работа алгоритма заканчивается, и W_i есть результат работы алгоритма. Если описанный процесс не заканчивается, то, по определению, алгоритм применим к слову P.

Определение 2.2. Словарная функция f в алфавите A (т.е. вида $f : A^* \to A^*$) называется вычислимой по Маркову, если существует схема нормального алгоритма S_a в алфавите $B \supseteq A$, вычисляющая

f, т.е. $f = F_s$.

Класс функций, вычислимых по Маркову, обозначим М.

Пример 2.3. $A = \{a_1, a_2\}$, схема

$$S_a: \begin{matrix} a_1 \to \cdot\lambda \\ a_2 \to \lambda \end{matrix}$$

Данный алгоритм оставляет пустое слово λ без изменения и всякое слово P в алфавите A переводит в слово Q, полученное из P путем вычеркивания первого вхождения буквы a_1. Алгоритм a неприменим к словам, не содержащим вхождений буквы a_1.

Пример 2.4. $A = \{a_1, a_2, ..., a_n\}$, схема

$$S_a: \begin{matrix} a_1 \to \lambda \\ a_2 \to \lambda \\ \dots \\ a_n \to \lambda \end{matrix}$$

Данный алгоритм переводит всякое слово P в алфавите A в пустое слово.

Пример 2.5. $A = \{a_1, a_2, ..., a_n\}$, схема

$$S_a: \lambda \to \cdot|$$

Данный алгоритм переводит всякое слово $P = \overbrace{||...|}^{x}$ в слово $Q = \underbrace{||...|}_{x+1}$. Если натуральное число n представить словом $|^{n+1}$, то данный алгоритм вычисляет функцию $f(n) = n + 1$.

Пример 2.6. $A = \{a_1, a_2, ..., a_n\}$, схема

$$S_a: \lambda \to \cdot\lambda$$

Данный алгоритм вычисляет функцию $F_s(P) = P$, $\forall P$. Если же взять схему $S_a: \lambda \to \lambda$, то такой алгоритм будет вычислять нигде не определимую функцию.

Пример 2.7. $A = \{a_1, a_2, ..., a_n\}$. Если $P = a_{i_1}, ..., a_{i_k}$, то обращением слова P называется слово $P = a_{i_k}, ..., a_{i_1}$.

Рассмотрим алфавит $B = A \cup \{\alpha, \beta\}$ и соответственно схему S_a.

1. $\alpha\alpha \to \beta$.
2. $\beta\alpha \to \alpha\beta, \forall a \in A$.
3. $\beta\alpha \to \beta$.
4. $\beta \to \cdot\lambda$.
5. $\alpha ab \to b\alpha a, \forall a, b \in A$.
6. $\lambda \to \alpha$.

Данный алгоритм осуществляет обращение слов в алфавите A.

В самом деле, пусть $P = a_{j_0}, ..., a_{j_k}$ – слово в алфавите A. Тогда

$$P \underset{(6)}{\to} \alpha P \underset{(5)}{\to} a_{j_1}\alpha a_{j_0}...a_{j_k} \underset{(5)}{\to} a_{j_1}a_{j_2}\alpha a_{j_0}a_{j_3}...a_{j_k} \underset{(5)}{\to} a_{j_1}a_{j_2}...a_{j_k}\alpha a_{j_0} \to$$

$$\underset{(6)}{\to} \alpha a_{j_1}a_{j_2}...a_{j_k}\alpha a_{j_0} \to ... \to a_{j_2}a_{j_3}...a_{j_k}\alpha a_{j_1}\alpha a_{j_0}.$$

Повторяя этот процесс, получим:

$$\alpha a_{j_k}\alpha a_{j_{k-1}}...\alpha a_{j_1}\alpha a_{j_0} \underset{(6)}{\to} \alpha\alpha a_{j_k}\alpha a_{j_{k-1}}...\alpha a_{j_1}\alpha a_{j_0} \to$$

$$\underset{(4)}{\to} \beta a_{j_k}\alpha a_{j_{k-1}}...\alpha a_{j_1}\alpha a_{j_0} \underset{(2)}{\to} a_{j_k}\beta\alpha a_{j_{k-1}}...\alpha a_{j_1}\alpha a_{j_0} \to$$

$$\underset{(3)}{\to} a_{j_k}\beta a_{j_{k-1}}...\alpha a_{j_1}\alpha a_{j_0} \underset{(2,3,4)}{\to} ... \to a_{j_k}a_{j_{k-1}}...a_{j_1}a_{j_0}.$$

Для нормальных алгоритмов разработана техника программирования, позволяющая осуществлять операции композиции алгоритмов, реализовывать операторы вида "если Φ, то выполнить F_1, иначе F_2", "пока Φ, выполнить F_1, иначе F_2".

Таким образом, класс функций M достаточно широк. В связи с этим А. Марковым был выдвинут принцип нормализации, который заключается в том, что все алгоритмы исчерпываются нормальными алгоритмами, другими словами – всякий алгоритм нормализуем. Принятие этого принципа позволяет истолковывать утверждения о несуществовании нормальных алгоритмов для решения конкретных задач как утверждения о несуществовании алгоритмов вообще. Данный принцип эквивалентен тезисам Чёрча и Тьюринга (см. подразд. 3.6 и 4.4).

Существуют сильные эвристические доводы в пользу того, что любой эффективный процесс над объектами некоторой формальной системы может быть описан в виде нормального алгоритма.

2.4. Челночные алгоритмы

Мы рассмотрим здесь некоторые разновидности марковских алгоритмов, обладающие рядом технических преимуществ. Эти разновидности будут введены в два этапа. На обоих этапах мы будем предполагать, что имеется класс вспомогательных букв, называемых челноками, таких, что каждое слово, над которым производится работа, содержит самое большее один из челноков и что эти челноки контролируют ход вычисления – в смысле, который мы сейчас объясним. Для обозначения челноков мы используем строчные греческие буквы; для других вспомогательных букв, которые играют роль указателей мест, будем использовать знаки препинания и другие аналогичные символы.

На первом этапе мы просто накладываем следующие условия:

(1) существует единственная команда, появляющаяся в конце алгоритма, такая, что ее левая часть пуста, а правая является челноком;

(2) любая другая команда содержит в левой части в точности один челнок;

(3) каждая команда, не являющаяся заключительной, содержит справа в точности один челнок;

(4) заключительная команда не содержит челнока справа.

Команда, фигурирующая в условии (1), будет называться начальной

командой, а челнок, имеющийся в ней,– начальным челноком. Алгоритм, для которого выполнены условия (1) – (4), будет называться получелночным алгоритмом.

Если у нас есть алгоритм, удовлетворяющий приведенным условиям, и мы начинаем со слова E, не содержащего челноков, то может применяться только начальная команда. После этого мы все время будем иметь слово в точности с одним челноком до тех пор, пока не достигнем заключительной команды; в этом случае процесс останавливается, а челноки исчезают.

Будем говорить, что команды с одним и тем же челноком слева принадлежат одной фазе. Алгоритм может быть построен так, что все команды, принадлежащие одной фазе, будут собраны вместе, а сами фазы можно располагать в произвольном порядке. Естественно ожидать, что различные фазы соответствуют важным подпроцессам в процессе, определяемом алгоритмом. Если, далее, каждая фаза заканчивается командой, левая часть которой состоит из одного лишь этого челнока, то алгоритм не сможет блокироваться.

Прежде чем идти дальше, остановимся, чтобы посмотреть, как описанный выше алгоритм удвоения может быть задан в виде получелночного алгоритма. В процессе удвоения существуют два основных этапа: копирование букв и сдвиг копий вправо. С этими двумя стадиями мы связали буквы α и β, но хотя α действует как челнок, β так не действует. Предположим теперь, что мы заменили в условии (2) β указателем места +. Когда α достигает правого конца выражения, пусть он превращается в челнок γ, который движется налево до тех пор, пока не встретит один из этих указателей места, а затем превращает этот указатель в β. Итак,

$$x\gamma \rightarrow \gamma x$$
$$+\gamma \rightarrow \beta$$

Затем команда (3) будет двигать βx (т. е. x, ранее отмеченное предшествовавшим знаком +) направо. По достижении βx правого конца β превращается в γ, и процесс продолжается до тех пор, пока γ не достигнет начала слова. Но в этом случае копия слова, которую мы должны иметь, будет обращенной. Чтобы предотвратить обращение, имеются два способа. Первый из них состоит в том, чтобы указатель

моста следующим образом восстанавливался:

$$\beta \to \gamma + \qquad (*)$$

Затем необходимо применить команды для выбрасывания этих указателей:

$$\gamma \to \beta$$
$$\delta x \to x\delta$$
$$\delta + \to \delta$$
$$\delta \to .$$

Второй способ состоит в замене α на $\gamma +$, а (*) – на

$$\beta x + \to \gamma + x .$$

Тогда мы сможем заключить алгоритм командой

$$\delta + \to .$$

Выгода этого способа состоит в том, что если мы пожелаем сохранить в результате + как разделительный знак (что часто бывает удобно), то мы можем совсем опустить δ-фазу. Весь алгоритм тогда будет иметь вид

$$\alpha x \to x + x\alpha$$
$$\alpha \to \gamma +$$
$$\beta xy \to y\beta x$$
$$\beta x + \to \gamma + x$$
$$x\gamma \to \gamma x$$
$$+\gamma \to \beta$$
$$\gamma \to .$$
$$\to \alpha$$

Как видно из приведенного примера, новый алгоритм содержит больше команд, нежели первоначальный. Но он удобен тем, что композиция алгоритмов упрощается и систематизируется.

На втором этапе мы примем следующие дополнительные условия (5): левая и правая части любой команды содержат самое большее одну букву, не являющуюся челноком; слева эта буква всегда появляется с одной и той же стороны челнока во всех командах одной фазы. Поэтому челнок оперирует с одной буквой, появляющейся всегда либо справа, либо слева от него; в первом случае челнок называется правосторонним, а во втором случае – левосторонним. Получелночный алгоритм, удовлетворяющий условию (5), и называется челночным алгоритмом.

Определение 2.3. **Челночный алгоритм** – это марковский алгоритм, в алфавит которого вводится специальный класс вспомогательных букв, называемых челноками, которые контролируют ход вычисления.

Для произвольного марковского алгоритма L можно найти получелночный алгоритм L', который с любым словом E действует так же, как L.

Челночный алгоритм является не менее общим, чем марковский. Но он может потребовать большего числа челноков, чем получелночный алгоритм.

2.5. Вычислимые функции

Рассмотрим некоторый данный счетно-бесконечный класс математических или логических вопросов, каждый из которых требует ответа "да" или "нет". Существует ли метод (процедура), с помощью которого мы можем ответить на любой вопрос из этого класса за конечное число шагов?

Говоря подробнее, мы интересуемся тем, можно ли для данного класса вопросов раз и навсегда описать процедуру или перечислить набор правил (предписаний), который можно было бы использовать следующим образом. Если (после того, как процедура описана) мы возьмем любой вопрос из данного класса, то процедура скажет нам, как выполнить последовательные шаги, после конечного числа которых мы получим ответ на рассматриваемый вопрос. При выполнении шагов нам придется только механически следовать предписаниям, как если бы мы были роботами: от нас не потребуется ни понимания, ни искусства, ни изобретательности. После каждого шага, если мы еще не получили окончательного ответа, предписания в сочетании с имеющейся ситуацией скажут нам, что делать дальше. Предписания позволят нам узнать, когда последовательность шагов заканчивается, и "считать" с получившейся в конце ситуации ответ на рассматриваемый вопрос: "да" или "нет"?

Поскольку никакой человек-исполнитель не может использовать более чем конечное количество информации, описание процедуры с помощью списка правил или предписаний должно быть конечным.

Если такая процедура существует, она называется разрешающей процедурой, или разрешающим алгоритмом для данного класса вопросов. Проблема разыскания разрешающей процедуры называется проблемой разрешения для этого класса.

Подобно тому как мы можем иметь разрешающую процедуру (алгоритм) для счетно-бесконечного класса вопросов, каждый из которых требует ответа "да" или "нет", мы можем иметь и вычислительную (или вычисляющую) процедуру (вычислительный алгоритм) для счетно-бесконечного класса вопросов, требующих в качестве ответа представления некоторого объекта. Проблема разыскания вычислительной процедуры (алгоритма) для класса вопросов типа "чему равно?" есть проблема вычисления для этого класса вопросов.

Мы сформулировали здесь понятия проблемы разрешения и проблемы вычисления только для счетно-бесконечных классов вопросов (типа "да или нет?" и "чему равно?" соответственно). Для конечного класса вопросов проблема разрешения или вычисления (сформулированная аналогично) с классической точки зрения тривиальна. Ведь по крайней мере теоретически ее можно было бы решить, просто приготовив список ответов на все вопросы из этого класса.

Возвращаясь к нашему случаю алгоритмов для счетно-бесконечных классов вопросов, заметим, что и они подобным же образом всегда существовали бы с классической точки зрения, если бы мы допустили, чтобы алгоритмы (описания алгоритмов) были бесконечными объектами; мы опять могли бы просто перечислить все ответы. Но мы сказали, что алгоритм должен быть методом (процедурой, набором правил), который может быть использован и который, следовательно, должен быть описан конечным образом. Должен быть дан конечный набор предписаний, достаточный, чтобы прийти к ответу на любой из бесконечного множества вопросов.

Поскольку наш класс вопросов всегда счетно-бесконечен, мы всегда можем перечислить все эти вопросы, скажем, в виде Q_0, Q_1, Q_2, ..., Q_a, ... Тогда переменной, или параметром, может быть a. В случае вопросов типа "да или нет?", полагая $P(a) \equiv$ {ответ на вопрос Q_a есть "да"}, мы превратим бесконечный класс вопросов типа "да или нет?" в одноместный теоретико-числовой предикат $P(a)$.

Предположим теперь, напротив, что рассматриваемые вопросы – это вопросы типа "чему равно?". Допустим далее, что объекты, которые должны быть предъявлены в ответах на вопросы, берутся из некоторого счетно-бесконечного класса, который мы можем перечислить в виде $y_0, y_1, y_2, ..., y_b, ...$ Полагая $f(a) = b$, если ответ на вопрос Q_a есть y_b, мы превратим бесконечный класс вопросов типа "чему равно?" в одноместную теоретико-числовую функцию $f(a)$.

Обратно, отправляясь от произвольного теоретико-числового предиката $P(a_1,...,a_n)$, мы получаем счетно-бесконечный класс вопросов: "Для данных $a_1,...,a_n$ истинно ли $P(a_1,...,a_n)$?". Отправляясь от произвольной теоретико-числовой функции $f(a_1,...,a_n)$, мы получаем вопросы: "Для данных $a_1,...,a_n$ чему равно значение $f(a_1,...,a_n)$?".

Таким образом, не имеет значения, говорим ли мы о счетно-бесконечных классах вопросов или о теоретико-числовых предикатах и функциях. Если для предиката (или для получающегося из него класса вопросов) существует разрешающая процедура, мы называем этот предикат (или класс вопросов) разрешимым. Подобным образом, если для некоторой функции имеется процедура вычисления, мы называем эту функцию вычислимой.

Определение 2.4. **Вычислимая функция** – функция $f(x)$, для которой существует алгоритм оценки для любого элемента x в области определения $f(x)$.

Осознание того, что для некоторых (счетно-бесконечных) классов математических вопросов у нас имеются алгоритмы, а для других они нам по меньшей мере неизвестны, относится к далекому прошлому математической истории. Греки, включая Евклида, искали алгоритмы. Напомним, что все проблемы вычисления и разрешения, как мы видели, могут быть сведены к проблемам вычисления для теоретико-числовых функций. Поэтому, чтобы продвинуться к нашей цели (к теореме Чёрча), нам нужно будет заняться вопросом: для каких теоретико-числовых функций имеются вычисляющие их процедуры (алгоритмы), т.е. каков класс "вычислимых" функций?

Чем мы располагаем сейчас, чтобы продолжить рассмотрение этого вопроса? Несколько расплывчатым интуитивным представлением о том, что представляет собой вычислительная процедура. Вычислительная

процедура может состоять просто в прямом применении определения функции или в чем-то совсем ином, что, как показывает математическая теория, должно вести к тем же значениям функции, которых требует исходное определение.

Хотя наше интуитивное понятие вычислительной процедуры расплывчато, оно тем не менее реально, как показывают следующие два обстоятельства.

Во-первых, оно не оставляет у математиков никаких сомнений и не приводит их ни к каким расхождениям в вопросе о том, что они располагают вычислительными процедурами для многих конкретных функций, например $a+1$, $a+b$, $a \cdot b$, a^b, $\max(a,b)$, $\min(a,b)$, $a \dot{-} 1$, $a \dot{-} b$, $\lfloor a/b \rfloor$, $\lfloor \sqrt{a} \rfloor$, $\lfloor e^a \rfloor$ и т. д.

Во-вторых, подобным же образом в других частных случаях нет сомнения в том, что определение некоторой функции или данный эквивалент ее определения не позволяет непосредственно указать вычислительную процедуру. Например, мы согласны с тем, что представленное выше определение функции $f(a)$ не дает для нее вычислительной процедуры.

Это интуитивное понятие вычислительной процедуры, которое достаточно реально, чтобы отделить те многие случаи, когда мы знаем, что имеем перед глазами вычислительную процедуру, от многих других случаев, когда мы знаем, что такой процедуры у нас нет, оказывается, однако, расплывчатым, когда мы пытаемся извлечь из него картину совокупности всех возможных вычислимых функций. А нам необходимо иметь такую картину, описанную в точных терминах, прежде чем мы сможем доказать, что для некоторой конкретной функции вообще не существует процедуры вычисления, или, короче, доказать, что некоторая функция невычислима. Для создания такой картины нужно нечто большее.

Большинство математиков-логиков сходятся на том, что это "нечто большее" было найдено Чёрчем (см. подразд. 3.6).

Начиная с этого места, мы будем иногда говорить, что нечто может быть сделано "эффективно", или что некоторая операция или процесс "эффективны", чтобы кратко выразить тот факт, что для этого имеется алгоритм (т. е. разрешающая или вычислительная процедура).

В связи с этим приведем строгие определения.

Определение 2.5. Последовательность преобразований для заданного элемента, позволяющая достичь заданной цели

преобразований за конечное число шагов, называется **эффективным процессом**.

Определение 2.6. **Описание эффективного процесса** – это описание, содержащее следующие условия: допустимость заданного элемента для данного процесса, описание преобразования и его результата, условие достижения цели.

Обычно предписание определяет эффективный процесс одновременно для многих элементов, т.е. процесс определен в общих терминах, включающих параметры.

Определение 2.7. **Эффективная процедура** – алгоритм, удовлетворяющий следующим требованиям: 1) он должен состоять из конечного множества простых (элементарных) команд, порядок исполнения которых должен быть однозначно задан; 2) при задании на вход алгоритма набора значений x из области определения $f(x)$ вычисления должны закончиться после конечного числа шагов и выдать соответствующее значение $f(x)$, а если набор значений x не принадлежит области определения $f(x)$, то вычисления не приводят ни к какому результату.

3. МАШИНА ТЬЮРИНГА И ВЫЧИСЛИМОСТЬ

3.1 Основы теории формальных грамматик

Понятие алгоритма используется в формальной лингвистике, в частности, следующим образом.

Выше мы определили (формальный) язык L над алфавитом A (над словарем B): L есть множество слов (фраз), принадлежащих свободной полугруппе A^* (соответственно B^*) с системой образующих A (соответственно B). Язык L представляет собой множество допустимых, т.е. (синтаксически) правильных слов (фраз). Поэтому $A^* \backslash L$ есть множество недопустимых, т.е. неправильных слов (фраз).

Формальная грамматика должна дать правила, которые обеспечивают либо получение правильно построенных слов (правильных фраз), либо распознавание того, является ли данная цепочка букв (слов) правильно построенным словом (правильной фразой). Другими словами, цель грамматики состоит в том, чтобы как-то задать (охарактеризовать) слова (фразы), принадлежащие данному языку. Таким образом, формальная грамматика предстает как алгоритм, а именно, как совокупность правил, позволяющих задать некоторый язык (точнее говоря, чтобы превратить грамматику в алгоритм, необходимо

присоединить к ней какой-либо механизм, устанавливающий жесткий порядок применения правил).

Предположим, что у нас имеется алгоритм такой, что его входными символами являются имена целых чисел 1, 2, ..., n и каждому входному символу отвечает на выходе некоторое слово (фраза) языка, определяемого данным алгоритмом. Тогда можно сказать, что этот алгоритм – или грамматика – перечисляет слова (фразы) этого языка.

Пример 3.1. Алфавит $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$; слово принадлежит языку L тогда и только тогда, когда оно является десятичным представлением простого числа.

Алгоритм, задающий язык L, легко себе представить; он может быть основан, например, на решете Эратосфена.

Пример 3.2. Алфавит $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$; язык L определяется следующим образом.

Рассмотрим (бесконечное) десятичное разложение числа π. Цепочка цифр является словом языка L тогда и только тогда, когда она встречается в этом разложении между двумя последовательными вхождениями символа 0. Алгоритм, задающий этот язык, основывается на вычислении числа n.

Естественным образом возникает следующая алгоритмическая проблема: по данной цепочке вхождений букв (слов) узнать, является ли она правильно построенным словом (правильной фразой) данного языка.

Пример 3.3. Всякий язык программирования состоит из исходных символов и правил образования выражений; правила приводятся в инструкции по использованию языка. (Если это хорошая инструкция, она представляет собой грамматику данного языка.) Правильная программа является правильной фразой языка, и обратно.

Компилирование программы, написанной на некотором языке программирования, для определенной конкретной машины – это операция, которую достаточно трудно определить формально. Грубо говоря, мы можем рассматривать компилирование программ как "перевод" с одного языка на другой: правильным цепочкам входного языка должны соответствовать правильные цепочки выходного языка. Процесс компилирования включает проверку синтаксической правильности входной программы. Эта проверка осуществляется специальным алгоритмом, имеющим два особых выходных символа, содержательный смысл которых – "правильно" и "неправильно".

Легко убедиться, что наличие алгоритма перечисления цепочек еще не гарантирует существования алгоритма распознавания их правильности. В примере 3.1 алгоритм, позволяющий узнать, является

ли данное число простым, существует. Однако в примере 3.2 дело обстоит иначе: по крайней мере, пока что никем не предложен алгоритм, позволяющий узнать, является ли данная цепочка словом из L. Быть может, когда-нибудь удастся доказать, что такой алгоритм вообще нельзя построить.

3.2. Регулярные языки и автоматные грамматики

Грамматики, все нетерминальные правила которых имеют вид

$$A \to aB;\ A, B \in V_A,\ a \in V_T,$$

называются автоматными грамматиками, сокращенно А-грамматиками (или finite-state grammars – грамматиками с конечным числом состояний).

Языки, порождаемые А-грамматиками, называются А-языками, или автоматными языками. Они были впервые введены и изучены С. К. Клини в связи с исследованием одной пробной модели нейрона; эти языки называются языками Клини, или автоматными языками (другие названия: регулярные языки, языки с конечным числом состояний).

Здесь мы будем рассматривать, полностью абстрагируясь от вероятностных подходов, только те общие свойства А-языков, которые связаны исключительно с "логическими" представлениями. Мы дадим ряд определений (или, скорее, вариантов определения) А-языков, а затем докажем их эквивалентность.

Хотя А-языки являются частным случаем КС-языков (контекстно свободных языков), представляется более интересным и полезным с методической точки зрения установить свойства А-языков независимо.

Автомат – это абстрактная кибернетическая машина, обрабатывающая входную последовательность символов согласно правилам порождающей грамматики и определяющая ее принадлежность некоторому формальному языку или выдающая некоторую выходную последовательность символов.

Определение 3.1. **Автоматная грамматика**, или А-грамматика, имеет алфавит $V = V_A \cup V_T$ (V_A – вспомогательный, V_T – основной), где $V_A \cap V_T = \varnothing$, $V_A = \{A_i \mid 1 \le i \le n\}$, причем A_1 – аксиома, $V_T = \{a_j \mid 1 \le j \le m\}$, и правила вида

$$A_i \to a_j A_k \quad \text{(правосторонние)}$$

или

$$A_i \to A_k a_j \quad \text{(левосторонние)}$$

(в каждой грамматике – только правосторонние или только левосторонние), а также вида

$$A_i \to a_j, \quad A_i \to E$$

(словарные или терминальные правила).

Стрелка в $X \to Y$ (как и выше) означает "подставить Y вместо X".

Пример 3.4. $V = \{P,Q\} \cup \{a,b\}$; аксиома – P;

$$\left(G_{\text{прав}}\right) : \left| \begin{array}{l} P \to aP \\ P \to aQ \\ Q \to bQ \\ Q \to b \end{array} \right|.$$

Эта грамматика задает язык, состоящий из всевозможных цепочек, устроенных следующим образом: последовательность из a, а за ней – последовательность из b, т. е.

$$\left\{a^m b^n \mid m,n \succ 0\right\}.$$

Тот же самый язык может быть описан другой грамматикой

$$\left(G_{\text{лев}}\right) : \left| \begin{array}{l} P \to Pb \\ P \to Qb \\ Q \to Qa \\ Q \to a \end{array} \right|.$$

Приведем в качестве примера вывод цепочки a^3b^2 в грамматике $\left(G_{\text{прав}}\right)$:

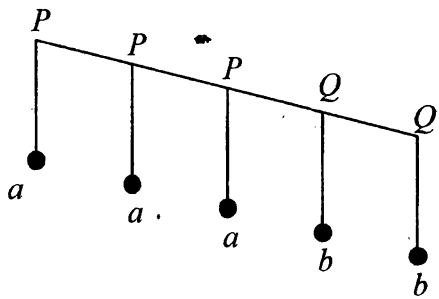
$$P \to aP \to aaP \to aaaQ \to aaabQ \to aaabb.$$

Замечание. Любой конечный язык является А-языком. Множество, содержащее n цепочек, – $\{f_1, f_2, ..., f_n\}$, $f_i = a_{i1}...a_{ik_i}$ – может быть описано следующей А-грамматикой:

$$G = \left\{P \to a_{i1}A_{i1}; A_{i2} \to a_{i2}A_{i2}; ...; A_{i,k_i-2} \to a_{i,k_i-1}A_{i,k_i-1}; A_{i,k_i-1} \to a_{ik_i} \mid 1 \le i \le n\right\}.$$

Синтаксические структуры. А-грамматика задает некоторое множество древовидных структур.

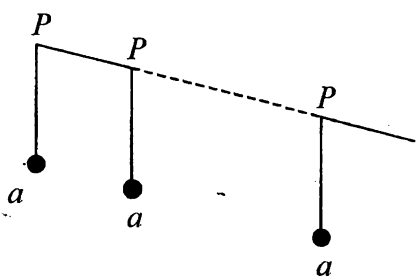
Пример 3.5. Цепочке *aaabb*, выведенной в $\left(G_{\text{прав}}\right)$, отвечает структура



В скобочной форме эта структура имела бы следующий вид:

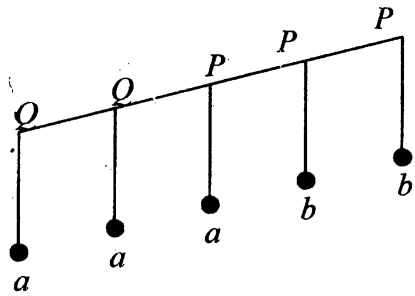
$$\underset{P}{(}a\underset{P}{(}a\underset{P}{(}a\underset{Q}{(}b\underset{Q}{(}b)))))$$

Структуру вида



иногда называют правоциклической.

Если бы для вывода цепочки мы воспользовались грамматикой $(G_{лев})$, то эта цепочка имела бы левоциклическую структуру



или в скобочной записи

$$\underset{PPPQQ}{((((a)a)a)b)b)}$$

Пример 3.6. Известный язык программирования Комит может быть описан посредством А-грамматики. Комит включает команды (правила) следующего вида:

NOM 1 Σ SYMB = Σ SYMB'//INST, ..., NOM 2,
левая часть правая часть

Здесь:

а) NOM 1 – имя (номер) данной команды;

б) Σ SYMB позволяет искать в памяти машины определенные последовательности символов, имеющие в Комите статус единой последовательности;

в) Σ SYMB' выполняет над символами, обнаруженными левой частью команды, определенные операции.

Таким образом, Σ SYMB содержит операнды, а Σ SYMB' – операции.

Формально Σ SYMB и Σ SYMB' суть последовательности любых слов, цифр или иных символов, разделенных знаком +. Например:

1 + МЯЧ + $ + СТАКАН + 2.

Команды Комита выполняются строго слева направо (именно этим обеспечивается "автоматность" языка);

г) INST обозначает некоторую последовательность команд, которая должна быть применена к результату работы правой части правила;

д) NOM 2 – имя (номер) команды, которая должна выполняться непосредственно вслед за данной.

Число элементов, фигурирующих в Σ SYMB и в Σ SYMB', ограничивается только реальным объемом памяти используемой ЭВМ. Однако на соотношения между Σ SYMB и Σ SYMB' могут быть наложены определенные ограничения. Для того чтобы Комит был А-языком, операции в Σ SYMB' должны быть определены независимо от операндов. Допустим, однако, что мы хотим, чтобы в ходе трансляции программы, написанной на Комите, можно было проверять, отвечает ли каждой операции в Σ SYMB' определенный операнд в Σ SYMB; тогда нам придется ввести ограничения, связывающие левую и правую части команды. Если подобные ограничения разрешается неограниченно вставлять друг в друга, то Комит уже не будет А-языком.

3.3. Конечные автоматы

А-грамматикам соответствуют машины весьма специального вида – так называемые конечные автоматы.

Конечный автомат – это простая разновидность автомата, которая однократно считывает входную строку слева направо, при этом в любой момент времени конечный автомат находится в некотором внутреннем состоянии, меняющемся после считывания очередного символа.

Формальный язык, распознаваемый конечным автоматом, называется **регулярным**.

Автоматные порождающие грамматики, позволяющие создать класс регулярных (автоматных) языков, относятся к наиболее простому типу грамматик.

Конечные автоматы допускают (распознают) все регулярные языки, порождаемые автоматными грамматиками, и только их.

Различают детерминированные и недетерминированные конечные автоматы.

Конечный автомат может принимать конечное число состояний $S_1, \ldots, S_p$. Он имеет читающую головку и ленту, которая перемещается под этой головкой (рис. 3.1). Лента разбита на ячейки; в каждой ячейке может находиться один символ из словаря $V = \{a_i \mid 0 \le i \le n\}$, где $a_0 = B$ играет роль пустого символа.

Одно из состояний автомата – S_0 – выделено и называется начальным состоянием.

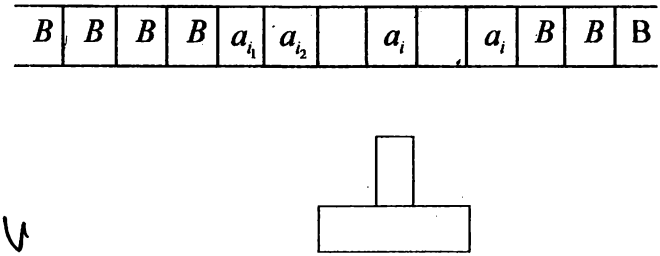


Рис. 3.1. Конечный автомат

На содержательном уровне функционирование конечного автомата можно представить себе следующим образом. На ленте записывается цепочка $\alpha \in V^*$; будем считать, что все ячейки слева и справа от α заполнены символами *B*. Автомат начинает работать в состоянии S_0, причем головка обозревает самый левый символ цепочки α. Прочтя обозреваемый символ, автомат переходит в новое состояние и перемещает ленту на одну ячейку влево. Переход в новое состояние выполняется в соответствии с одной из команд, имеющих вид

$$\left(a_i, S_j\right) \to S_k$$

Число команд конечно; левая часть $\left(a_i, S_j\right)$ называется ситуацией автомата, а правая часть (S_k) есть состояние, в котором автомат будет находиться на следующем шаге (после выполнения данной команды).

Если автомат оказывается в ситуации $\left(a_{i_1}, S_{j_1}\right)$, не являющейся левой частью какой-либо команды, то он останавливается. Если после проведения вычислений над цепочкой α, автомат возвращается к ситуации $\left(B, S_0\right)$, говорят, что автомат допускает цепочку α. Множество цепочек, допускаемых данным автоматом, есть, по определению, язык, допускаемый этим автоматом. Класс языков, допускаемых описанным классом, совпадает с классом А-языков.

Пример 3.7. Вот множество команд конечного автомата, допускающего язык $\left\{a^m b^n \mid m, n \succ 0\right\}$:

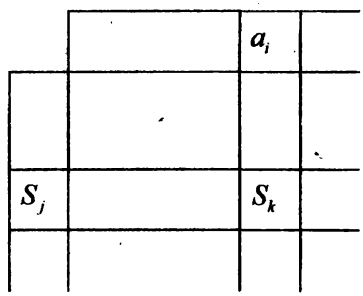
$$(a,S_0)\to S_1$$
$$(a,S_1)\to S_1$$
$$(b,S_1)\to S_2$$
$$(b,S_2)\to S_2$$
$$(B,S_2)\to S_0$$

В литературе встречаются различные варианты определения конечного автомата. Основные из них связаны со следующими модификациями.

Пустой символ можно использовать в качестве маркера, а можно и вообще отказаться от него.

Вместо того чтобы в конце вычислений возвращаться в начальное состояние S_0, автомат может принимать одно из выделенных заключительных состояний; в частности, он может иметь единственное заключительное состояние.

Переходы автомата из одного состояния в другое могут описываться не командами описанного выше вида, а матрицей переходов, или же функцией переходов. (Ясно, впрочем, что это различие относится лишь к способу записи.)



Пример 3.8. Язык $\{a^m b^n\}$ может быть задан посредством следующей матрицы:

	a	b	B
S_0	S_1		
S_1	S_1	S_2	
S_2		S_2	S_0

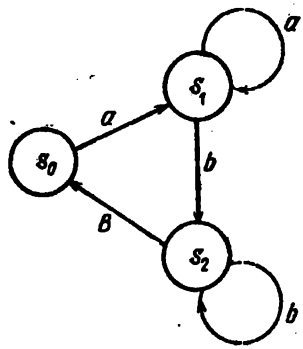
Функцию переходов мы будем обозначать через $M\left(a_i,S_j\right)$. Она определена на подмножестве декартова произведения $V\times S$ и принимает значения из S для случая детерминированного автомата, который мы здесь рассматриваем; если же автомат недетерминированный, то значениями функции M являются подмножества S.

Мы будем доопределять M на множестве $V^*\times S$ следующим образом. Пусть $\varphi\in V^*$ и $\varphi=\varphi_1 a_i$; тогда

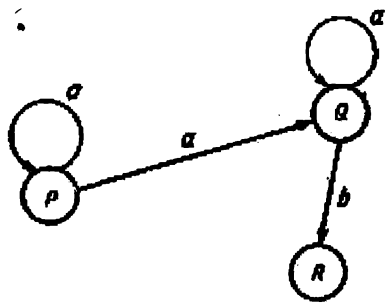
$$M\left(a_i,S_j\right)=M\left(a_i,M\left(\varphi_1,S_j\right)\right).$$

Произвольному конечному автомату можно поставить в соответствие некоторый "размеченный" граф, действуя следующим образом. Каждому состоянию автомата сопоставляется вершина графа, каждой команде вида $\left(a_i,S_j\right)\to S_k$ сопоставляется дуга, ведущая из S_j в S_k и помеченная символом a_i.

Пример 3.9. Автомату из предыдущих примеров отвечает следующий граф:



Аналогичным образом можно сопоставить граф А-грамматике: нетерминальным символам ставятся в соответствие вершины графа, каждому правилу вида $A_i \to a_j A_k$ – дуга с пометкой a_j, ведущая из A_i в A_k; каждому правилу вида $A_i \to a_j$ отвечает дуга с пометкой a_j, ведущая из A_i в особую вершину *R*, не отвечающую никакому нетерминальному символу. Так, для грамматики $\left(G_{\text{прав}}\right)$ в примере 3.5 получается следующий граф:



От графа, отвечающего конечному автомату, граф А-грамматики отличается тем, что в первом невозможно, чтобы две дуги, выходящие из одной и той же вершины *A* и помеченные одним и тем же символом a, входили в разные вершины *B* и *C*, тогда как для второго графа такого ограничения нет. Строго говоря, и автомату, и грамматике ставится в соответствие не граф, а мультиграф.

3.4. Машина Тьюринга

Один из способов формализации понятий вычислимости и алгоритма состоит в рассмотрении некоторой идеализированной вычислительной машины. При этом, поскольку мы хотим иметь возможность высказывать о ней как можно больше математических утверждений, она должна быть как можно более простой. Поэтому наша идеализированная машина не будет резко отличаться по своему строению от реальных электронных вычислительных машин (ЭВМ).

Концепция абстрактной вычислительной машины была предложена еще в 1936 г. (т.е. до создания первых электронных вычислительных машин) английским математиком и логиком **Аланом Тьюрингом**. Он показал возможность существования универсальной вычислительной машины (машины Тьюринга), способной выполнить любую эффективную процедуру.

Машина Тьюринга – это гипотетическая вычислительная машина (автомат), использовавшаяся в качестве математической абстракции А. Тьюрингом с целью уточнения эффективного процесса (алгоритма) вычислений.

Если говорить об абстрактной схеме построения машины Тьюринга, то она представляет собой хорошую математическую модель электронно-вычислительной машины.

Верно, что у машины Тьюринга предполагается потенциально бесконечная память; однако если ЭВМ обладает внешней памятью на магнитной ленте или на дисках, причем мы располагаем достаточным количеством бобин ленты или запасом дисков, то ситуация оказывается практически такой же.

Верно, что машина Тьюринга читает за один ход лишь один символ; однако этот символ может иметь сложное значение.

Наконец, как и арифметическое устройство ЭВМ, машина Тьюринга имеет конечное число внутренних состояний. Ее поведение в каждый данный момент зависит от состояния, в котором она находится, и от того символа, который она получает в качестве внешней информации. Если отвлечься от того, что процесс ввода информации идеализирован и сведен к чтению одного атомарного символа на каждом шаге, то поведение машины Тьюринга не отличается от функционирования реальной ЭВМ.

Машина Тьюринга, по определению, представляет собой совокупность нескольких компонент, которые мы опишем сейчас по отдельности. При этом мы будем, следуя установившейся традиции, пользоваться наглядными образами, так что по форме наше

определение будет не вполне строгим. Однако без использования этих образов изложение проиграло бы в ясности и простоте. На содержательном уровне машина Тьюринга состоит из управляющего устройства, входной ленты и головки, которая в отличие от головки конечного автомата может не только считывать символы с ленты, но и записывать на нее новые символы (рис. 3.2).

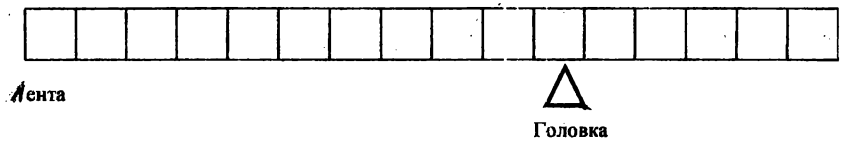


Рис. 3.2. Машина Тьюринга

С точки зрения формального описания машина Тьюринга есть совокупность следующих пяти компонентов:

1) управляющего устройства, способного принимать некоторое число состояний, которые соответствуют различным комбинациям состояний внутренних элементов электронной машины (ячейки памяти, электронные схемы и т.п.);

2) ленты, разделенной на конечное число ячеек (≥ 1). На ленте выбрано направление, называемое направлением слева направо. Соответственно этому на ленте выделяются крайняя слева и крайняя справа ячейки (в частном случае, когда лента состоит из одной ячейки, они совпадают); каждая ячейка, кроме крайней слева, имеет единственную соседнюю слева, каждая, кроме крайней справа, имеет единственную соседнюю справа. В процессе работы машины лента может изменяться, а именно, к ней могут подклеиваться новые ячейки справа (но не слева!), т.е. число ячеек предполагается неограниченным в обе стороны;

3) головки, которая в каждый момент работы машины находится в некоторой ячейке ленты и считывает информацию из этой ячейки. Головка может работать в каждый момент только с одной ячейкой ленты. При этом она может заменять прочитанный символ другим символом;

4) двух конечных алфавитов – внешнего

$$\Sigma = \{a_{-1}, a_0, ..., a_n\}, \quad n \geq 1$$

и внутреннего

$$S = \{q_0, q_1, ..., q_l\},$$

называемого также множеством состояний.

Внешний алфавит – это конечный алфавит символов, которые подаются на вход машины Тьюринга и выдаются на ее выходе.

Внутренний алфавит – это конечный алфавит символов, комбинации которых определяют внутреннее состояние машины Тьюринга.

Элементы Σ называются внешними символами машины, элементы S – внутренними состояниями или просто состояниями. В каждый момент работы машина находится в некотором состоянии (т. е. ей сопоставлен некоторый элемент алфавита S) и в каждой ячейке ленты записан некоторый (единственный) внешний символ. Таким образом, внешние символы видны на ленте, являющейся внешним устройством машины, в то время как состояние не видно и является в каждый момент чем-то внутренне присущим машине в целом.

Символы a_{-1}, a_0, q_0, q_1 играют особую роль и носят особые названия: a_{-1} называется граничным маркером, a_0 – пустым символом, q_0 – заключительным состоянием, q_1 – начальным состоянием. Особенности употребления этих символов будут точно указаны ниже, предварительно же можно сделать следующие пояснения:

- граничный маркер записывается в крайней слева ячейке и остается в ней в течение всего процесса вычисления, чтобы головка не могла "свалиться" с ленты влево (сваливание вправо предотвращается подклеиванием ячейки). Граничный маркер в процессе вычисления не может быть ни стерт, ни записан заново в какую-либо ячейку. Из ячейки, в которой записан граничный маркер, головка не может сдвинуться влево;
- вместо a_{-1} мы часто будем писать #;
- пустой символ служит для того, чтобы можно было рассматривать ленту, у которой в некоторых ячейках ничего не записано (это удобно из содержательных соображений) и в то же время формально считать, что в каждой ячейке всегда записан какой-то символ (что упрощает формулировки и рассуждения). Таким образом, если в некоторой ячейке записан символ a_0, содержательно это означает, что ячейка пуста;
- начальное и заключительное состояния указывают соответственно на начало и конец процесса вычисления;
- вместо a_1 мы часто будем писать "1" или "|";

• символы П и Л будут означать смещение ленты на одну ячейку вправо или влево соответственно.

Дадим теперь несколько определений.

Определение 3.2. *Выражением* называется конечная последовательность символов в алфавите $\{a_1, a_0, a_1, ..., q_1, q_2, ..., П, Л\}$.

Определение 3.3. *Командой* называется выражение одного из трех следующих типов:

$$q_i a_j a_k q_m$$
$$q_i a_j П q_m ;$$
$$q_i a_j Л q_m$$

5) программы машины – конечного множества слов специального вида, называемых командами.

Среди вычислений машины нам удобно будет выделить некоторые "правильные", которых для всех ваших целей будет достаточно. Правильными мы будем считать такие вычисления, при которых:

• в крайней слева ячейке ленты записан граничный маркер, и ни в какой другой ячейке его нет;

• в начальный и заключительный моменты головка обозревает крайнюю слева ячейку;

• в начальный момент машина находится в начальном состоянии, а в заключительный момент – в заключительном;

• в заключительный момент все пустые ячейки (если они есть) расположены правее всех непустых.

Теперь мы определим правильное вычисление формально.

Определение 3.4. Правильным вычислением машины Тьюринга М мы будем называть последовательность $(s_0, s_1, ..., s_p)$ конфигураций этой машины такую, что:

для каждого $i = 0, ... p-1$ ситуация s_{i+1} получается из s_i выполнением некоторой команды машины М;

s_0 – начальная конфигурация;

s_p – заключительная конфигурация.

Ситуацией машины называется состояние машины вместе с указанием, какая ячейка обозревается и всего того, что напечатано на ленте.

Формализацией машины Тьюринга называется набор $M = \langle \Sigma, S, \Pi, q_0, q_l, a_0 \rangle$, где Σ – внешний алфавит, S – внутренний алфавит, Π – программа машины, q_0 – начальное состояние, q_l – заключительное состояние, a_0 – пробел (бланк).

Работа машины Тьюринга. Машина Тьюринга работает следующим образом: в каждой ситуации, определенной внутренним состоянием управляющего устройства и символом, прочитанным на ленте, машина срабатывает, переходя в другое состояние и передвигая ленту определенным образом. Это делается в соответствии с правилами перехода, которые указывают, как должна действовать машина в каждой заданной ситуации.

С помощью только что приведенных определений работа машины Тьюринга может быть теперь описана точно.

Начальная пара некоторой команды задает исходную ситуацию: машина находится в состоянии q_i и читает символ a_j. Поведение машины в данной ситуации $q_i a_j$ полностью определяется заключительной парой символов той единственной команды, которая соответствует этой ситуации (т. е. начинается парой символов $q_i a_j$). При этом элементарные действия машины бывают одного из следующих трех типов:

- тип $a_k q_m$: машина заменяет a_j на a_k и переходит из состояния q_i в состояние q_m;
- тип Πq_m: машина сдвигает ленту на одну ячейку вправо и переходит из состояния q_i в состояние q_m;
- тип $Л q_m$: машина сдвигает ленту на одну ячейку влево и переходит из состояния q_i в состояние q_m.

Напомним, что на множество команд, задающих машину Тьюринга, наложено требование, чтобы начальные пары символов всех команд были различны. В силу этого требования в любой данной ситуации возможно не более чем одно очередное элементарное действие. Машину, для которой это требование выполнено, мы назовем

детерминированной. Машину, для которой указанное требование не соблюдается, т. е. в некоторой ситуации она имеет право "выбирать" между несколькими разными элементарными действиями, мы будем называть недетерминированной.

Сформулируем еще одно определение.

Определение 3.5. *Конфигурацией* называется выражение, удовлетворяющее следующим условиям:

- оно не содержит вхождений символов П и Л;
- оно содержит ровно одно вхождение символа типа q_i;
- это вхождение символа q_i не занимает в выражении крайней правой позиции.

Так, выражения

$$1q_3a_4Ba_2BBa_1$$

и

$$q_3B1$$

являются конфигурациями, а выражения

$$Лq_1,\ 1a_2,\ q_2Ba_3q_3a_1,\ a_1a_2q_1$$

таковыми не являются.

Пусть α – некоторая конфигурация. Будем говорить, что α есть конфигурация некоторой машины Тьюринга М, если состояние q_i, фигурирующее в α, является одним из внутренних состояний машины М, а все остальные символы, входящие в α, принадлежат алфавиту М.

Конфигурация машины однозначно задает ее поведение. В самом деле, характеристика ситуации не учитывает всю совокупность символов, записанных на ленте; конфигурация же позволяет сделать это.

Конфигурация, из которой выброшено q_i, характеризует содержимое ленты; в то же время, если a_j – символ, стоящий в конфигурации справа от q_i, то q_ia_j есть ситуация машины М.

Дадим еще одно определение.

Определение 3.6. Пусть α и β – конфигурации некоторой машины М. Мы будем писать

$$\alpha \to \beta$$

в одном из трех следующих случаев (P и Q обозначают произвольные последовательности символов алфавита машины):

- $\exists P,Q: \alpha = Pq_i a_j Q$, $\beta = Pq_m a_k Q$, машина имеет команду $q_i a_j a_k q_m$;
- $\exists P,Q: \alpha = Pq_i a_j Q$, $\beta = Pa_j q_m Q$, машина имеет команду $q_i a_j Л q_m$;
- $\exists P,Q: \alpha = Pa_k q_i a_j Q$, $\beta = Pq_m a_k a_j Q$, машина имеет команду $q_i a_i П q_m$.

Первый случай соответствует такому элементарному действию машины, при котором она заменяет символ a_j символом a_k и переходит из состояния q_i в состояние q_m.

Второй случай соответствует действию, состоящему в передвижении ленты на одну ячейку влево и переходе из состояния q_i в состояние q_m.

Третий случай соответствует передвижению ленты на одну ячейку вправо и переходу из состояния q_i в состояние q_m.

Из определения 3.6 непосредственно следует, что если $\alpha \to \beta$ и $\alpha \to \gamma$, то $\beta = \gamma$ (здесь используется детерминированность); если М и М'–две машины Тьюринга, такие, что всякая команда машины М есть команда машины М' (в этом случае пишут $M \subset M'$), то из $\alpha \to \beta$ относительно М следует $\alpha \to \beta$ и относительно М'.

Конфигурация α называется заключительной относительно машины М, если не существует конфигурации β этой машины, такой, что $\alpha \to \beta$.

Определение 3.7. Назовем вычислением машины Тьюринга М последовательность конфигураций

$$\alpha_1, \alpha_2, \ldots, \alpha_p$$

таких, что $\alpha_i \to \alpha_{i+1}$ $(1 \le i \prec p)$ и α_p – заключительная конфигурация (относительно М). Будем писать

$$\alpha_p = \text{Рез}M(\alpha_1)$$

что означает: "α_p есть результат переработки α_1 машиной М".

Условимся считать, что в общем случае машина перерабатывает слово, записанное в алфавите $\{a_i\}$; при этом a_0 обозначает пустой символ и исходное слово окружено пустыми символами с обеих сторон. Машина располагает таким количеством пустых ячеек, которое необходимо ей для вычислений. Одно из состояний машины принимается за начальное; машина начинает работу, находясь в этом состоянии, причем головка установлена перед самым левым (непустым) символом, записанным на ленте.

Приведем примеры машин Тьюринга.

Пример 3.10. Пусть машина М имеет алфавит $A = \{a_0, a_1, a_2\}$ и состояния q_1, q_2, q_3; состояние q_1 является начальным. Поведение машины описывается следующими командами:

$$q_1 a_0 Л q_1 \quad (1)$$
$$q_1 a_2 Л q_1 \quad (2)$$
$$q_1 a_1 Л q_2 \quad (3)$$
$$q_2 a_1 П q_3 \quad (4)$$
$$q_2 a_0 Л q_1 \quad (5)$$
$$q_2 a_2 Л q_1 \quad (6)$$
$$q_3 a_1 a_2 q_3 \quad (7)$$

Пояснения:

а) если машина находится в начальном состояли q_1 и читает символ a_0 или a_2, то она сдвигает ленту на одну ячейку влево, оставаясь в состоянии q_1; теперь головка обозревает символ, стоящий в ячейке

непосредственно справа от a_0 или a_2 (команды (1) и (2));

б) если машина находится в состоянии q_1 и читает символ a_1, то она сдвигает ленту на одну ячейку влево и в результате начинает обозревать символ, соседний справа к a_1. При этом она "запоминает" тот факт, что видела a_1: состояние q_2 можно интерпретировать как информацию о том, что был прочитан символ a_1 (команда (3));

в) если, все еще "помня об a_1", машина видит a_0 или a_2, она забывает, что видела a_1 (возврат в состояние q_1), и переходит к обозреванию символа, стоящего непосредственно справа от a_0 или a_2 (команды (5) и (6));

г) если, напротив, находясь в состоянии q_2 ("помня об a_1"), машина снова видит a_1, она возвращается к предыдущему символу (это как раз тот a_1, который привел ее в состояние q_2) и переходит в состояние q_3 (команда (4)). Затем она заменяет первый из этих двух соседних символов a_1 на a_2 и переходит в состояние q_3 (команда (7)). Теперь машина оказывается в ситуации q_3a_2, для которой нет подходящей команды; поэтому она останавливается;

д) таким образом, единственным фактором, влекущим остановку машины, является наличие двух последовательно написанных a_1. Если в процессе чтения входного слова машина не встретит двух рядом стоящих a_1, то она дойдет до конца слова, перейдет на пустые ячейки справа от него и будет работать вечно (команда (1)).

Зададим машине М, находящейся в состоянии q_1, слово

$$a_1a_0a_2a_1a_1a_0a_1.$$

В результате получится последовательность конфигураций:

$$
\begin{array}{lll}
q_1\,a_1\;\; a_0\;\; a_2\;\; a_1\;\; a_1\,a_0\,a_1 & (1)\\
a_1\;q_2\;a_0\;\; a_2\;\; a_1\;\; a_1\,a_0\,a_1 & (2)\\
a_1\;\; a_0\;q_1\;a_2\;\; a_1\;\; a_1\,a_0\,a_1 & (3)\\
a_1\;\; a_0\;\; a_2\;q_1\;a_1\;\; a_1\,a_0\,a_1 & (4)\\
a_1\;\; a_0\;\; a_2\;\; a_1\;q_2\;a_1\,a_0\,a_1 & (5)\\
a_1\;\; a_0\;\; a_2\;q_3\;a_1\;\; a_1\,a_0\,a_1 & (6)\\
a_1\;\; a_0\;\; a_2\;q_3\;a_1\;\; a_1\,a_0\,a_1 & (7)
\end{array}
$$

((7) – заключительная конфигурация).

Теперь предложим той же машине слово

$$a_1a_2a_2.$$

В этом случае вычисления пойдут так:

$$
\begin{array}{lll}
q_1\,a_1\;\; a_2\;\; a_2 & (1)\\
a_1\;q_2\;a_2\;\; a_2 & (2)\\
a_1\;\; a_2\;q_1\;a_2 & (3)\\
a_1\;\; a_2\;\; a_2\;q_1\;a_0 & (4)\\
a_1\;\; a_2\;\; a_2\;\; a_0\;q_1\;a_0 & (5)
\end{array}
$$

Процесс продолжается вечно, т.е. результат здесь не определен.

Пример 3.11. Транскрибирующая машина.

На ленте машины Тьюринга записано слово в алфавите $A=\{a_1,...,a_n\}$; слева и справа его окружают пустые ячейки #.

Пусть имеется другой алфавит $B=\{b_1,...,b_n\}$, находящийся с первым во взаимно однозначном соответствии: $a_i \leftrightarrow b_i$.

Задача машины Т состоит в транскрибировании любого данного A-слова с помощью алфавита B, причем транскрибированное слово должно быть записано на другом участке ленты, а исходное сохранено.

В процессе работы машина использует маркер m. Вот совокупность команд, или программа, определяющая машину Т.

1. Для всякого i:

1.1. $q_0 a_i m q_i$.

Находясь в начальном состоянии q_0 и прочитав a_i – самую правую букву исходного слова, машина пишет вместо a_i маркер m и запоминает a_i посредством перехода в состояние q_i .

1.2. $q_i m \Pi q_i$.

Находясь в состоянии q_i (т. е. помня о символе a_i) и видя маркер m, Т перемещает ленту вправо, оставаясь в состоянии q_i .

1.3. $q_i \# a_i q_l^i$.

Находясь в состоянии q_i и видя пустую ячейку, машина Т записывает в нее a_i и переходит в состояние q_l^i : "прочитанный перед этим символ a_i скопирован".

1.4. $q_l^i m \Lambda q_l^i$.

Видя маркер после копирования a_i (не обязательно сразу), машина сдвигает ленту влево.

2. Для всякой пары $\langle i, j \rangle$:

2.1. $q_i a_j \Lambda q_i$.

2.2. $q_l^i a_j \Lambda q_l^i$.

2.3. $q_l^i b_j \Lambda q_l^i$.

Скопировав a_i и находясь в состоянии q_l^i , машина вне зависимости от обозреваемой буквы передвигает ленту влево; при этом остается в состоянии q_l^i .

2.4. $q_l^i \# b_i r$.

Дойдя до пустой ячейки, Т транскрибирует a_i, вписывая b_i в пустую ячейку, после чего переходит в состояние r – "состояние возврата".

3. Для всякого j:

3.1. $rb_j \Pi r$.

3.2. $ra_j \Pi r$.

Когда машина находится в состоянии r, то, какова бы ни была обозреваемая буква, Т остается в состоянии r и сдвигает ленту вправо.

Машина Т продолжает сдвигать ленту вправо до тех пор, пока не наткнется на маркер, оставленный ею там, откуда она "отправилась" искать свободные места для копирования и транскрибирования a_i. Тогда она стирает маркер, оставаясь в состоянии r.

3.3. $r \# \Lambda q_0$.

Видя пустой символ, которым она только что заменила маркер, машина сдвигает ленту влево (так, что теперь она будет обозревать очередной символ исходного слова) и возвращается в начальное состояние q_0.

Когда Т закончит транскрибирование, она окажется в состоянии q_0, но сможет прочесть только какую-либо из букв b_j. Поскольку ситуации $\langle q_0, b_j \rangle$ не соответствует ни одна команда, машина остановится.

В этот момент на ленте будут записаны оба слова (исходное A-слово и его B-транскрипция), разделенные пустой ячейкой.

3.5. Вычислимость по Тьюрингу

С помощью понятия машины Тьюринга формализуется интуитивное понятие вычислимости (а тем самым и интуитивное понятие алгоритма). Мы имеем в виду прежде всего психологическое представление о вычислимости: вычислителю (обычно человеку) предлагаются некоторые исходные данные и сообщаются правила их обработки; он должен

получить тот или иной результат, зависящий от этих данных. Нередко вычислитель-человек может заранее сказать, вычислим ли требуемый результат: ведь, как правило, он обладает определенным практическим опытом вычислительной работы. Так, известно, что:

- число π не вычислимо, поскольку его десятичное представление бесконечно;
- приближенное значение числа π с недостатком вычислимо с точностью до 10^{-n} при любом фиксированном n.

Однако в ряде случаев неизвестно, является ли требуемый результат вычислимым.

Заметим, что, говоря о вычислимости, мы отвлекаемся от естественных (физических) ограничений, характерных как для человека, так и для машины: для некоторых вычислений может понадобиться больше ячеек памяти, чем имеется элементарных частиц во всей нашей Галактике, и время, измеряемое миллиардами тысячелетий; тем не менее, невзирая на материальную неосуществимость подобных вычислений, можно видеть, что они представляют эффективный способ получить некоторый результат за конечное число шагов.

Были предприняты некоторые математические исследования с целью выработать более реалистическое понятие вычислимости; однако, как бы то ни было, математические теории, имеющие дело с вычислимостью, не занимаются практическими вычислениями, такими, например, с какими имеет дело численный анализ.

Таким образом, представление о вычислимости, формализуемое с помощью точного понятия машины Тьюринга, является очень широким; мы увидим тем не менее, что даже в рамках этого широкого понятия некоторые объекты, которые интуитивно кажутся "вычислимыми", на самом деле таковыми не являются.

Описанные в подразд. 3.4 машины Тьюринга позволяют реализовать алгоритмы самого общего характера. В частности, два предшествующих примера относились к "нечисленным" алгоритмам. Однако, по правде говоря, на уровне машин Тьюринга между "численным" и "нечисленным" нет существенного различия (для реальных вычислительных машин это различие, напротив, с практической точки зрения весьма важно; пригодность машины для той или иной области приложений зависит от набора "встроенных" операций: машины для количественных расчетов нуждаются в разнообразных арифметических операциях, машины для логической обработки информации – в булевых операциях, операциях типа подстановки и т. д.). Когда речь идет просто о манипуляциях с символами, мы вольны давать этим символам любую интерпретацию –

и числовую, и нечисловую. Если мы хотим проводить вычисления, т.е. работать с числами, то их можно изображать многими разными способами.

Для работы с целыми неотрицательными числами мы будем пользоваться абстрактным двухбуквенным алфавитом $A = \{a_0, a_1\}$: a_0 соответствует пустому символу, обозначаемому B; a_1 соответствует палочке, обозначаемой |.

Всякое целое неотрицательное число n изображается последовательностью из $n + 1$ палочек. Это соглашение позволяет отличать число "нуль", изображаемое палочкой, от пустого символа, играющего роль разделителя. Мы будем обозначать подобное изображение числа n через $\overline{n}$; например:

$$\overline{0} = |,\ \overline{1} = ||,\ \ldots,\ \overline{4} = |||||.$$

Конечная последовательность $n_1, n_2, \ldots, n_k$ будет изображаться следующим образом:

$$\overline{n}_1 B \overline{n}_2 \ldots B \overline{n}_k .$$

Пусть имеется выражение M, иначе говоря, цепочка в алфавите $\{B, |; q_1, q_2, \ldots q_n; П, Л\}$. Будем обозначать через $\langle M \rangle$ число палочек, которое она содержит.

Пусть М – машина Тьюринга с состояниями $q_1, q_2, \ldots$. Каждой n-ке $(m_1, \ldots, m_n)$ неотрицательных целых чисел мы поставим в соответствие конфигурацию

$$\alpha_1 = q_1 \overline{m}_1 B \overline{m}_2 B \ldots B \overline{m}_n .$$

Если для машины М существует вычисление, начинающееся конфигурацией α_1 и доходящее до заключительной конфигурации α_p:

$$\alpha_1 \to \alpha_2 \to \ldots \to \alpha_p ,$$

то число $\langle \alpha_p \rangle$ есть функция от М и начальной n-ки; мы будем писать

$$\langle \alpha_p \rangle = \Psi_Z^{(n)}(m_1, ..., m_n).$$

Если же вычисления, начинающегося с α_1, не существует, т. е. если не существует целого числа p, такого, что конфигурация α_p является заключительной, то функция $\Psi_Z^{(n)}$ не определена для рассматриваемой n-ки.

Исходя из машины М, мы пришли к функции с целыми неотрицательными значениями, определенной на множестве N^n или, быть может, на некотором подмножестве множества N^n (частично определенная функция) (здесь и далее N означает множество натуральных чисел, включая нуль, а N^n – множество всевозможных упорядоченных n-ок натуральных чисел).

Поступим теперь наоборот: попробуем исходить из функций, определенных на N^n или его подмножестве.

Определение 3.8. Говорят, что функция f, определенная на некотором подмножестве множества N^n, является **частично вычислимой**, если существует машина Тьюринга М такая, что для всякой n-ки $(x_1, ..., x_n)$, которой отвечает некоторое значение f, выполняется равенство

$$f(x_1, ..., x_n) = \Psi_Z^{(n)}(x_1, ..., x_n).$$

Функция f называется вычислимой, если она определена на N^n и является частично вычислимой. Чаще встречается такое определение: частичную функцию называют вычислимой, если существует машина Тьюринга М такая, что для всякой n-ки $(x_1, ..., x_n)$, которой отвечает значение f, выполняется равенство

$$f(x_1, ..., x_n) = \Psi_Z^{(n)}(x_1, ..., x_n),$$

а для тех n-ок, для которых значения f не существует, машина работает вечно.

Определение 3.9. Функция называется **вычислимой по Тьюрингу**, если существует машина Тьюринга, вычисляющая эту функцию.

Практическая вычислимость – это вычислимость, которая реализуется с помощью физического автомата и является значительно более узким понятием, чем вычислимость по Тьюрингу.

В дальнейшем нам понадобится также понятие вычислимого предиката. Его нельзя ввести просто как частный случай вычислимой функции, так как значения предиката – предложения – не являются конструктивными объектами. Поэтому мы примем следующее естественное определение.

Определение 3.10. Предикат $P(x_1,...,x_n)$, определенный на некотором подмножестве n-й декартовой степени множества всех слов в некотором конечном алфавите, называется вычислимым (по Тьюрингу), если вычислима функция, сопоставляющая каждому набору $x_1^0,...,x_n^0$ значений временных $x_1,...,x_n$ истинностное значение предложения $P(x_1,...,x_n)$.

Пример 3.12 (пример вычислимой функции).

Функция $f(x,y)=x+y$, определенная на множестве пар неотрицательных целых чисел, является "вычислимой в интуитивном смысле"; покажем, что она вычислима и в смысле Тьюринга. Для этого построим машину Тьюринга М с алфавитом $\{B, |\}$ такую, что

$$x+y=\Psi_Z^{(n)}(x,y).$$

Подобная машина должна, имея на ленте коды $\overline{x}$ и $\overline{y}$, разделенные пустой ячейкой, получать из этой записи $x+y$ палочек (не обязательно подряд). Заметим, что

$$x+y=\langle\overline{x}\rangle+\langle\overline{y}\rangle-2.$$

Легко убедиться, что следующая машина выполняет поставленную задачу:

$q_1|Bq_2$ (1) Стирается первая палочка первого кода.

$q_2 B Л q_3$ (2)
$q_3 \mid Л q_3$ (3)
$q_3 B Л q_4$ (4)

Лента сдвигается до тех пор, пока головка не дойдет до первой палочки второго кода.

$q_4 \mid B q_4$ (5) Стирается первая палочка второго кода.

Здесь машина останавливается, поскольку ситуации $q_4 B$ нет ни в одной команде. В этот момент на ленте ровно $x + y$ палочек.

Пример 3.13 (пример частично вычислимой функции).

Функция $g(x, y) = x - y$ определена лишь для тех пар, у которых $x \geq y$. Покажем, что она частично вычислима в смысле Тьюринга.

Для этого мы построим машину Тьюринга М, выполняющую вычитание. Пусть на ленте имеются коды $\overline{x}$ и $\overline{y}$, разделенные пустой ячейкой: $\overline{x}$ слева, $\overline{y}$ справа. Машина будет работать последовательными циклами; каждый цикл состоит в стирании одной (самой левой) палочки в $\overline{x}$ и одной (самой правой) палочки в $\overline{y}$. Вот соответствующая программа.

$q_1 \mid B q_1$ (1) Стирается самая левая палочка в $\overline{x}$.

$q_1 B Л q_2$ (2) Лента сдвигается на одну ячейку влево.

$q_2 \mid Л q_2$ (3) Лента сдвигается влево до тех пор, пока головка не дойдет до конца кода $\overline{x}$.

$q_2 B Л q_3$ (4) Головка переходит через пробел (пустую ячейку), разделяющий $\overline{x}$ и $\overline{y}$.

$q_3 \mid Л q_3$ (5) Лента сдвигается влево до тех пор, пока головка не дойдет до конца кода $\overline{y}$.

$q_3 B П q_4$ (6) Попав на пустую ячейку, примыкающую к $\overline{y}$ справа, головка возвращается на одну ячейку назад (т. е. к самой правой палочке $\overline{y}$).

$q_4 \mid B q_4$ (7) Самая правая палочка в $\overline{y}$ стирается.

$q_4 B П q_5$ (8) Лента сдвигается на одну ячейку вправо, и машина переходит в "проверочное" состояние.

Дело в том, что продолжение или прекращение вычисления зависит от результатов некоторых проверок.

$q_5 | П q_6$ (9) Если обозреваемая ячейка содержит палочку ($\overline{y}$ еще не стерт полностью), вычисление продолжается. Если же обозреваемая ячейка пуста, то, поскольку ситуации $q_5 B$ не отвечает никакая команда, машина останавливается.

$q_6 | П q_6$ (10) Лента сдвигается вправо до тех пор, пока головка не дойдет до начала кода $\overline{y}$.

$q_6 B П q_7$ (11) Головка переходит через пробел, разделяющий $\overline{y}$ и $\overline{x}$.

$q_7 | П q_8$ (12) Если в $\overline{x}$ осталась хотя бы одна палочка, вычисление продолжается.

$q_7 B П q_7$ (13) Если в $\overline{x}$ не осталось ни одной палочки (случай $x \prec y$), машина будет работать вечно, сдвигая ленту вправо.

$q_8 | П q_8$ (14) Лента сдвигается вправо до тех пор, пока головка не дойдет до начала того, что осталось от кода $\overline{x}$.

$q_8 B Л q_1$ (15) Головка устанавливается так, что обозревается самая левая палочка "остатка" кода $\overline{x}$; машина переходит в начальное состояние q_1. Тем самым она оказывается готовой начать очередной цикл.

Частично вычислимую функцию $x - y$ нетрудно тривиальным образом продолжить до вычислимой функции. Из этого, разумеется, нельзя делать вывод, что всякую частично вычислимую функцию можно продолжить до вычислимой.

3.6. Тезис Черча

Рассмотрим теперь доводы в пользу того, что понятие машины Тьюринга действительно может служить уточнением интуитивного понятия алгоритма – иначе говоря, в пользу следующего утверждения.

137.

Тезис Черча: всякая эффективно вычислимая функция является вычислимой по Тьюрингу.

Другими словами, для всякой функции, которую можно каким-либо способом вычислить, можно построить вычисляющую ее машину Тьюринга. Это утверждение, называемое обычно тезисом Черча – Тьюринга, впервые было высказано А. Тьюрингом в 1936 г. применительно к конструкции машины, от которой описанная выше отличается лишь несущественными деталями. (Сходная конструкция была предложена тогда же Э. Постом, который является также автором еще одной конструкции, предназначенной для той же цели, известной под именем продукций Поста). Аналогичное утверждение применительно к некоторому совсем другому уточнению того же интуитивного понятия было несколько раньше высказано А. Черчем.

Тезис Черча важен для приложений математики в естествознание. При этом необходимо отметить, что тезис Черча не является математическим утверждением, так как в его формулировку входит понятие алгоритма в интуитивном смысле этого слова.

По своему статусу этот тезис похож скорее на законы физики, представляющие собой обычно утверждения о том, что то или иное явление природы описывается некоторой математической конструкцией (например, дифференциальным уравнением) – с той разницей, что речь идет не о явлении природы, а об одном из видов человеческой деятельности – вычислении. В справедливости физических законов мы убеждаемся путем анализа опытных данных; аналогичным образом к убеждению в справедливости тезиса Черча-Тьюринга приводит нас анализ фактов, относящихся к вычислительной практике и смежным с ней областям.

138

Обратимся прежде всего к вычислению, выполняемому человеком без использования вспомогательных средств. Слово "вычисление" мы будем понимать в широком смысле – не обязательно как оперирование с числами. Примерами "нечисловых вычислений" могут служить тождественные преобразования в элементарной алгебре или в логике предложений, нахождение производных от элементарных функций и т. п. Каждое вычисление состоит в последовательном преобразовании некоторого комплекса символов, записанных на бумаге или на доске. (Вычисления "в уме" можно не рассматривать, так как их всегда можно заменить письменными вычислениями, имеющими ту же структуру.) Принципиально ничего не изменится, если мы будем представлять себе, что всякое вычисление производится на одной достаточно большой доске (доска отличается от бумаги тем, что написанное на ней можно стирать и на освободившемся месте писать снова) и что доска

разграфлена на клетки, причем в каждой клетке можно записать только один символ. Вычисление можно теперь рассматривать как последовательность шагов, на каждом из которых изменяется содержимое одной клетки; в общем случае это означает, что записанный в клетке символ стирается и на его месте записывается другой. Но от чего зависит, в какой клетке производится очередной шаг и что в ней на этом шаге записывается? Разумеется, не только от того, что было раньше записано в ней самой, но и от содержимого других клеток; например, при сложении "столбиком" записываемая в клетке цифра определяется цифрами, стоящими над ней. Следовательно, прежде чем изменить содержимое какой-либо клетки, нужно, вообще говоря, просмотреть некоторые другие клетки и запомнить какие-то сведения о том, что в них находится. Кроме того, следующий шаг может зависеть от предыдущих; поэтому после выполнения очередного шага бывает нужно что-то о нем запомнить.

Мы можем сказать теперь, что кроме "явных" шагов, на которых изменяется содержимое клеток, имеются еще "неявные", состоящие в просмотре клеток без изменения их содержимого. Но между этими двумя типами шагов нет принципиальной разницы, так как просмотр клетки можно представлять себе как "нулевое изменение": записываемый в клетке символ совпадает с находившимся там ранее. Таким образом, вычисление состоит из "элементарных шагов", на каждом из которых изменяется содержимое какой-то клетки, и обо всяком шаге вычислитель должен, вообще говоря, что-то запомнить. В любой момент он держит в памяти какие-то сведения об уже выполненной части вычисления. После очередного шага к этим сведениям что-то добавляется и вместе с тем что-то из имевшихся ранее сведений может быть на данном шаге "использовано до конца" и за ненужностью забыто; естественно сказать, что при каждом шаге "состояние памяти" изменяется. Это "состояние памяти" можно представлять себе как набор сведений о некоторых уже выполненных шагах, причем для каждого шага указываются, скажем, координаты клетки, где он выполняется, характер изменения содержимого и "давность" этого шага по отношению к настоящему моменту. Все эти или любые другие сведения о выполненных шагах можно записать в виде некоторого слова в конечном алфавите (для данного вычисления фиксированном). Но объем памяти человека ограничен: существует константа (сейчас нам неважно, какая), ограничивающая сверху длины запоминаемых слов. Поэтому число возможных "состояний памяти" конечно.

Заметим теперь, что в отличие от "явных шагов", с которых мы начинали анализ процесса вычисления, каждый "элементарный шаг"

зависит только от состояния памяти и от содержимого той клетки, где этот шаг выполняется (результаты просмотра других клеток учитываются в состоянии памяти). На каждом "элементарном шаге" изменяются содержимое одной клетки и состояние памяти (разумеется, изменение состояния памяти также может быть нулевым). Кроме того, каждый предыдущий "элементарный шаг" должен включать в себя указание "адреса" клетки, где нужно произвести следующий. Не будет существенным ограничением считать, что эта новая клетка, если она не совпадает со старой, расположена рядом с ней – ведь до любой клетки можно дойти "мелкими шажками", сдвигаясь каждый раз на одну клетку (вправо, влево, вверх или вниз).

Итак, мы расчленили процесс вычисления на "элементарные шаги", каждый из которых состоит в том, что "вычислитель", обозревая некоторую клетку и учитывая, каково состояние его памяти и что записано в обозреваемой клетке, производит три операции: 1) заменяет символ, записанный в обозреваемой клетке, другим символом; 2) переходит к обозрению одной из четырех соседних клеток или продолжает обозревать ту же клетку; 3) изменяет состояние памяти. Такой шаг может быть выполнен часто механически, без понимания его смысла, если имеется инструкция – при таком-то состоянии памяти и таком-то символе в обозреваемой клетке заменить этот символ на такой-то, сдвинуться в такую-то сторону (или не сдвигаться) и заменить состояние памяти на такое-то. Эта инструкция может быть, очевидно, записана в точно такой же форме, в какой записываются команды машины Тьюринга: $qa \to a'Cq'$, где q и q' – состояния памяти, a и a' – символы, используемые при вычислении, и C принимает одно из значений "вправо", "влево", "вверх", "вниз", "на месте". Итак, анализ процесса вычисления приводит к "двумерной машине Тьюринга", отличающейся от рассматривавшейся одномерной только тем, что лента заменена разграфленной на клетки доской и головка движется не в двух, а в четырех направлениях. Такая двумерная машина вполне могла бы использоваться в качестве уточнения понятия алгоритма. Но одномерная машина "технически проще" и поэтому удобнее для формальной трактовки (т. е. для доказательства теорем и т. п.), хотя принципиальной разницы между одномерным и двумерным вариантами нет: клетки, на которые разграфлена доска, можно тем или иным регулярным способом расположить на одномерной ленте. При этом клетки, соседние на доске, не обязательно будут соседними на ленте, но всегда можно "найти" на ленте клетку, которая на доске была рядом с обозреваемой, с помощью некоторой "стандартной подпрограммы".

Таким образом, анализ того, как вычисляет человек, подтверждает тезис Черча. Применение вспомогательных средств (например, таблиц) или механических приспособлений (счеты, арифмометр) не дает ничего принципиально нового: использование таблиц всего лишь увеличивает объем памяти вычислителя (что может быть очень важно практически, но теоретического значения не имеет), а вычисления с помощью простых механизмов можно "моделировать" последовательными изменениями их (механизмов) схем, выполненных с помощью символов на клетчатой доске.

Что касается вычислений с помощью современных вычислительных машин с программным управлением, то и их принципиальные возможности, как известно, не превышают возможностей человека-вычислителя (единственное их преимущество – быстродействие). Кроме того, читатель, хотя бы немного знакомый с этими машинами, наверняка сразу заметил, что машины Тьюринга очень на них похожи, и построение машин Тьюринга для вычисления конкретных функций – это, в сущности, программирование (в отличие от программы для ЭВМ программа машины Тьюринга считается частью самой машины, но это вопрос терминологии). Главное отличие машин Тьюринга от реальных ЭВМ (если отвлечься от ограниченности объема памяти и времени работы последних) – крайняя простота элементарных шагов; эта простота позволяет легко строить "трансляторы" с языков реальных машин на язык машины Тьюринга. Можно сказать даже, что машина Тьюринга – это общая абстрактная схема вычислительной машины с программным управлением (но стоит подчеркнуть, что Тьюринг придумал свои машины за несколько лет до появления первых ЭВМ).

Подытоживая все изложенное, можно сказать, что анализ строения реальных вычислительных процедур убеждает нас в возможности указать для каждой такой процедуры осуществляющую ее машину Тьюринга. Это и есть главный довод в пользу тезиса Черча; к нему можно добавить еще два косвенных довода:

1) для уточнения понятия алгоритма предложено довольно много конструкций, основанных на различных соображениях и весьма разнообразных по форме; нередко в двух таких конструкциях трудно на первый взгляд усмотреть что-либо общее. Однако все эти конструкции равносильны между собой в том смысле, что всякая функция, вычислимая с помощью какой-либо одной из них, вычислима и с помощью любой другой. (При этом для каждой пары конструкций их равносильность удается строго доказать.) Никто еще не смог предложить конструкцию, приводящую к более широкому классу вычислимых функций;

2) до сих пор во всех случаях, когда для какой-либо конкретной функции, вычислимость которой в интуитивном смысле не вызывает сомнений, исследовался вопрос, вычислима ли она также и в смысле какого-либо уточнения понятия алгоритма (а значит, и всех известных уточнений), на этот вопрос удавалось получить положительный ответ.

Тезисом Черча часто пользуются для сокращения доказательств теорем теории алгоритмов: если по ходу рассуждения нужно установить, что некоторая функция вычислима по Тьюрингу, довольствуются указанием "неформальной" вычислительной процедуры и построение машины Тьюринга опускают.

Проблема остановки – проблема, возникающая после запуска машины Тьюринга, когда могут возникнуть две ситуации: машина может остановиться через конечное число шагов или будет работать бесконечно.

О проблеме остановки известно, что она не разрешима.

4. РЕКУРСИВНЫЕ МНОЖЕСТВА И ФУНКЦИИ

4.1. Понятие рекурсии

Здесь и далее речь пойдет об уточнении понятия вычислимой функции, в основе которого лежит широко распространенный способ вычисления функций натурального аргумента, состоящий в том, что задается $f(0)$ и так называемая рекуррентная (от лат. recurro – возвращаюсь) формула $f(n+1)=h(f(n))$, позволяющая вычислить $f(n+1)$, если известно $f(n)$, и с помощью этой формулы последовательно вычисляются $f(1)$, $f(2)$ и т. д.

Такой процесс последовательного вычисления значений функции называется *рекурсией*. Метод рекурсии дает возможность сводить вычисление одной функции к вычислению другой: если мы умеем вычислять h, то мы сумеем вычислить и любое значение f. Примером может служить функция 2^n, которую можно задать равенствами $2^0=1, 2^{n+1}=2\cdot 2^n$, сводящими вычисление ее значений к вычислению значений функции $h(m)=2m$. Таким образом, если из вычислимой (в интуитивном смысле) функции получена с помощью рекурсии другая функция, то эта новая функция также вычислима (в том же смысле).

Разумеется, рекурсия – не единственный способ получать одни

функции из других, при котором вычислимая функция переходит в вычислимую. Тем же свойством обладает, например, подстановка (суперпозиция): если мы умеем вычислять функции $f(x)$ и $g(x)$, то мы сумеем вычислить и функцию $g(f(x))$.

Если у нас есть некоторый запас простейших функций, вычисление которых тривиально, то все функции, которые можно из них получить, производя сколько угодно раз суперпозиции и рекурсии, с точки зрения любой разумной интуиции также будут вычислимыми.

Естественно спросить: насколько широк класс вычислимых функций, которые можно так получить? Прежде чем выяснять этот вопрос, нужно, разумеется, уточнить, из какого запаса простейших функций мы исходим. Но целесообразно также обобщить понятие рекурсии – и даже в двух направлениях.

Во-первых, $f(n+1)$ может зависеть не только от $f(n)$, но и от самого n; в этом случае рекуррентная формула имеет вид $f(n+1)=h(n,f(n))$, где h – некоторая заведомо вычислимая функция двух переменных.

Пример 4.1. Пусть $f(n)=n!$, тогда $h(k,m)=m(k+1)$, что дает формулу $(n+1)!=n!(n+1)$. (В рассмотренном выше частном случае, когда h – одноместная функция и $f(n+1)=h(f(n))$, функцию f часто называют итерацией (от лат. iteratio – повторение) функции h).

Во-вторых, с помощью рекурсии можно вычислять также и функции нескольких переменных. При этом все переменные, кроме одной, служат параметрами, и для каждого фиксированного набора их значений вычисление функции от оставшейся переменной производится точно так же, как это делалось для одноместных функций. Это значит, что если мы вычисляем функцию $f(x_1,...,x_k)$, $k>1$, и рекурсия проводится по переменной x_k, то $f(x_1,...,x_{k-1},0)$ вычисляется по формуле $f(x_1,...,x_{k-1},0)=g(x_1,...,x_{k-1})$, а $f(x_1,...,x_{k-1},x_k+1)$ – по формуле $f(x_1,...,x_{k-1},x_k+1)=h(x_1,...,x_{k-1},x_k,f(x_1,...,x_{k-1},x_k))$, где g и h – заведомо вычислимые функции вместимости $k-1$ и $k+1$ соответственно.

Пример 4.2. Пусть $f(x_1, x_2) = x_1 \cdot x_2$, тогда функция $g(x)$ тождественно равна нулю, а $h(x_1, x_2, x_3)$ равна $x_3 + x_1$, что дает формулы $x_1 \cdot 0 = 0$, $x_1 \cdot (x_2 + 1) = x_1 \cdot x_2 + x_1$.

Рекурсивные функции используются в вычислительной математике для изучения эффективно вычислимых функций.

4.2. Рекурсивные и рекурсивно перечислимые множества и предикаты

Задачи о нахождении алгоритмов для вычисления функций (в частности, предикатов) называют обычно алгоритмическими проблемами; если алгоритма для вычисления той или иной функции не существует, говорят, что соответствующая алгоритмическая проблема неразрешима.

Утверждения о неразрешимости проблем распознавания свойств имеют следующий вид: "Не существует алгоритма, позволяющего по любому конструктивному объекту из заданного класса А узнать, обладает ли этот объект данным свойством". Тезис Черча позволяет уточнить эту формулировку так: "Не существует машины Тьюринга, распознающей данное свойство объектов класса А". При этом предполагается, что класс А состоит из слов в некотором конечном алфавите; но в действительности очень часто речь идет о классах иной природы, элементы которых кодируются словами. Что касается выражения "машина Тьюринга распознает свойство", то оно понимается так: машина перерабатывает каждый элемент класса А в "и", если он обладает данным свойством, и в "л" – если не обладает (вместо "и" и "л" можно, использовать любые другие символы.). Аналогичным образом уточняются утверждения о нераспознаваемости отношений.

Проще всего доказывается неразрешимость некоторых алгоритмических проблем, относящихся к самой теории алгоритмов, а именно, проблем распознавания определенных свойств машин Тьюринга. Из них самая простая – проблема распознавания самоприменимости. Машина Тьюринга называется самоприменимой, если она применима к своему собственному коду. В противном случае машина называется несамоприменимой.

Теорема 4.1. Не существует алгоритма, позволяющего для любой машины Тьюринга узнать, является ли она самоприменимой.

В соответствии со сказанным выше точный смысл этого утверждения таков: не существует машины Тьюринга, перерабатывающей

код любой самоприменимой машины Тьюринга в "и" и код любой несамоприменимой в "л".

Доказательство. Допустим, что такая машина существует; обозначим ее M_0. По этой машине легко построить другую машину M_1, работающую на кодах несамоприменимых машин так же, как M_0, а на кодах самоприменимых машин работающую вечно.

Какова бы ни была машина Тьюринга *M*, машина M_1 применима к ее коду тогда и только тогда, когда *M* несамоприменима. Но отсюда следует, что, во-первых, M_1 не может быть самоприменимой, так как тогда она была бы неприменима к своему коду, т.е. была бы несамоприменима; во-вторых, M_1 не может быть несамоприменимой, так как в этом случае она была бы применима к своему коду, т.е. была бы самоприменима. Противоречие доказывает теорему.

Точное понятие алгоритма позволяет выделить среди всевозможных функций вычислимые, т.е. такие, значения которых можно "в самом деле находить". Под таким же углом зрения можно рассмотреть и понятие множества: весьма естественно попытаться среди всевозможных способов задания множеств выделить "эффективные", т.е. такие, которые позволяли бы "в самом деле работать" с множествами – указывать их конкретные элементы и производить над ними вычислительные операции, отвечать для конкретных объектов на вопрос, входят ли эти объекты в данное множество, и т. п. При этом, разумеется, придется ограничиться множествами, состоящими из конструктивных объектов; как и прежде, моделью конструктивного объекта будет служить слово в конечном алфавите. Однако во многих случаях удобно вместо слов рассматривать натуральные числа, пользуясь тем, что между словами в заданном конечном алфавите и натуральными числами можно установить очень простое взаимно однозначное соответствие. Именно, если фиксировать произвольное целое $P \succ 1$, то любое целое положительное *n* может быть единственным образом представлено в виде

$$n = \alpha_1 P^{k-1} + \alpha_2 P^{k-2} + \ldots + \alpha_{k-1} P + \alpha_k , \qquad (4.1)$$

где α_i принимают значения $1,\ldots,P$ (доказательство по индукции). Поэтому, если выбрать конечный алфавит $V = \{a_1,\ldots,a_p\}$ мощности P и

сопоставить числу n, представленному в виде (4.1), слово $a_{\alpha_1}a_{\alpha_2}...a_{\alpha_k}$, то каждому целому положительному n будет отвечать единственное слово; обратно, для каждого непустого слова $a_{\alpha_1}a_{\alpha_2}...a_{\alpha_k}$ в алфавите V по формуле (4.1) вычисляется единственное целое положительное число, которому отвечает это слово. Если, кроме того, числу 0 сопоставить пустое слово, мы получим взаимно однозначное соответствие между натуральным рядом и множеством всех слов в алфавите V. Число, соответствующее слову x, мы будем обозначать νx и называть стандартным номером этого слова, а само отображение ν – стандартной нумерацией слов в алфавите V.

От обычной записи натуральных чисел в системе счисления с основанием P представление с помощью отображения ν отличается тем, что "цифры" $a_1a_2...a_p$ имеют значения $1,...,P$, а не $0,..,P-1$. Для наших целей обычная запись непригодна из-за того, что при ней используются не все слова, составленные из цифр, а только те, которые не начинаются с нуля.

Здесь мы будем все время, говоря о множествах, подразумевать множества натуральных чисел (и говоря о дополнении множества – дополнение до натурального ряда). Кроме того, будем считать, что фиксирован некоторый конечный алфавит $V=\{a_1,...,a_p\}$, $P\succ 1$, и во всех вычислениях на машинах Тьюринга числа представляются словами в этом алфавите описанным только что способом.

Существуют, как известно, два основных способа задания множества – путем указания характеристического свойства его элементов и путем их перечисления. Владея точным понятием алгоритма, мы можем указать для каждого из этих двух способов "алгоритмический вариант". Для первого способа такой вариант получается наложением условия, чтобы характеристическое свойство было распознаваемым (или, как часто говорят, *разрешимым*), т.е. чтобы существовал алгоритм, позволяющий по любому слову узнать, обладает ли оно этим свойством. В алгоритмическом варианте второго способа перечисление должно быть "в самом деле перечислением", т.е. происходить с помощью некоторого перечисляющего алгоритма.

Эти соображения нетрудно уточнить следующим образом. В первом варианте не вполне четкое понятие "характеристическое свойство" можно заменить понятием "характеристический предикат". Во втором варианте перечисляющий алгоритм можно понимать как машину Тьюринга, которая

"выдает" в заключительных ситуациях все возможные элементы множества, если "подавать" на ее ленту в начальных ситуациях все возможные натуральные числа. Иначе говоря, перечисляемое множество должно быть множеством значений (одноместной) функции, которую вычисляет некоторая машина Тьюринга с входным и выходным алфавитами, совпадающими с нашим алфавитом V. Это приводит к следующим двум определениям.

Определение 4.1. **Словарный предикат** – выражение $P(x_1,...,x_k)$, где x_i – слова в некотором алфавите. Например, x_1 пробегает слова в алфавите Δ_1, x_2 – слова в алфавите Δ_2 и т.д. Про такой предикат говорят, что он относится к типу $\Delta_1^* \times \Delta_2^* \times ... \times \Delta_k^*$, указывая область пробегания каждого аргументного места.

Определение 4.2. **Характеристическая функция предиката *P*** – функция вида $\chi_P : \Delta_1^* \times \Delta_2^* \times ... \times \Delta_k^* \to \{0,1\}$ такая, что для всех наборов $\langle x_1,...,x_k \rangle$, где $x_i \in \Delta_i^* : P(x_1,...,x_k)$ истинно $\Leftrightarrow \chi_P(x_1,...,x_k) = 1$ ($\Leftrightarrow$ – тогда и только тогда, когда...).

Определение 4.3. **Рекурсивный предикат** – предикат P типа $\Delta_1^* \times \Delta_2^* \times ... \times \Delta_k^*$, для которого вычислима по Тьюрингу соответствующая характеристическая функция χ_P.

Определение 4.4. **Предикат принадлежности множества *X*** – предикат, для которого выполняется следующее: $P(x)$ истинно $\Leftrightarrow x \in X$, при этом задано множество слов $X \subseteq \Sigma^*$ и всякое слово $x \in \Sigma^*$.

Определение 4.5. **Рекурсивное множество** – это множество $X \subseteq \Sigma^*$, для которого рекурсивен соответствующий предикат принадлежности $x \in X$ и где x пробегает множество Σ^*.

Рекурсивными множествами являются многие обычные множества натуральных чисел: множество четных чисел, множество простых чисел и т. д.

Определение 4.6. Множество называется рекурсивно перечислимым, если оно является множеством значений некоторой вычислимой функции.

Определение 4.6′. **Рекурсивно перечислимый предикат** – предикат P рекурсивно перечислим, если существует процедура, позволяющая устанавливать истинность $P(x_1,...,x_k)$; если же $P(x_1,...,x_k)$ ложно, то эта процедура иногда будет это устанавливать, а иногда будет продолжаться неограниченно долго.

Теорема 4.2. Применение логических связок $\wedge$, $\vee$, $\neg$ к рекурсивным предикатам дает рекурсивные же предикаты.

Теорема 4.2′. Применение квантора к рекурсивному предикату приводит к рекурсивному же предикату.

Интуитивно предикат P рекурсивен, если существует алгоритм, позволяющий выяснить для каждого набора $\{x_1,...,x_k\}$ слов истинно $P(x_1,...,x_k)$ или нет.

Обычно арифметические предикаты $x \le y$, $x + y = z$, "x – простое число", "x делит y" являются рекурсивными.

Здесь и далее под вычислимыми функциями будут подразумеваться одноместные вычислимые функции.

Рассмотрим свойства рекурсивных и перечислимых множеств.

Теорема 4.3. Объединение и пересечение рекурсивных множеств рекурсивны; дополнение к рекурсивному множеству рекурсивно.

Ввиду известного соответствия между теоретико-множественными операциями и пропозициональными связками эта теорема немедленно вытекает из следующей теоремы.

Теорема 4.3′. Дизъюнкция и конъюнкция вычислимых предикатов вычислимы; отрицание вычислимого предиката вычислимо.

Теорема 4.3′ справедлива для предикатов любой вместимости; мы, однако, докажем ее только для одноместных предикатов – этого достаточно, чтобы получить теорему 4.3. Для общего случая теорема доказывается аналогично.

Доказательство теоремы 4.3′.

1. Пусть машины Тьюринга M_1 и M_2 вычисляют предикаты $F_1(x)$ и $F_2(x)$ соответственно. Тогда машина, вычисляющая предикат $F_1(x) \cup F_2(x)$ (с входным алфавитом, равным объединению входных алфавитов M_1 и M_2), будет работать следующим образом. Если на ее

ленте в начальной ситуации записано слово (число) x, она это слово удваивает, вставляя между двумя его экземплярами разделительный знак (не принадлежащий объединению внешних алфавитов M_1 и M_2). Затем она работает со вторым экземпляром x как M_1 (обращаясь при этом с разделительным знаком как с граничным маркером); если в результате получится И (ИСТИНА), то все содержимое ленты стирается и вслед за (настоящим) граничным маркером записывается и; если же в результате работы со вторым экземпляром x получилось Л (ЛОЖЬ), то машина стирает это л и разделительный знак, а затем работает с первым экземпляром x как M_2.

2. Если машина M вычисляет предикат $F(x)$, то машина, вычисляющая $\neg F(x)$ (с тем же входным алфавитом), должна работать так же, как M, до самого конца, а затем, если получилось И, переработать его в Л, а если Л – то переработать его в И.

3. Утверждение для конъюнкции следует из утверждений для дизъюнкции и отрицания ввиду соотношения $F_1(x) \cap F_2(x) \equiv \neg\left(\neg F_1(x) \cup \neg F_2(x)\right)$.

Теорема 4.4. Объединение и перечисление перечислимых множеств перечислимы.

Доказательство. Пусть машины M_1 и M_2 вычисляют функции $f_1(x)$ и $f_2(x)$ с множествами значений E_1, и E_2 соответственно. Тогда:

1) по машинам M_1 и M_2 легко построить машину, которая будет вычислять функцию f, определенную следующим образом:

$$f(2x) = f_1(x),\ f(2x+1) = f_2(x).$$

Значениями f будут те и только те числа, которые служат значениями хотя бы одной из функций f_1, f_2, так что множество ее значений есть $E_1 \cup E_2$;

2) нетрудно построить по M_1 и M_2 также машину, вычисляющую функцию g, такую, что: а) для чисел вида $x = 2^{x_1} \cdot 3^{x_2}$ $g(x)$ определена

тогда и только тогда, когда $f_1(x_1)$ и $f_2(x_2)$ определены и совпадают, и в этом случае значение $g(x)$ равно общему значению $f_1(x_1)$ и $f_2(x_2)$; б) для остальных x значение $g(x)$ не определено. Значениями g будут те и только те числа, которые служат значениями как f_1, так и f_2, так что множество ее значений есть $E_1 \cap E_2$.

Теорема 4.5. Множество тогда и только тогда перечислимо, когда оно является областью определения некоторой вычислимой функции.

Доказательство.

1. Пусть машина *M* вычисляет функцию $f(x)$ с множеством значений *E*. Рассмотрим машину, входной и выходной алфавиты которой совпадают с выходным алфавитом машины *M*, работающую следующим образом. Если в начальной ситуации на ее ленте записано число *x*, машина ставит справа от него разделительный знак ("квадратик") и справа от этого знака проделывает два первых шага того вычисления машины *M*, которое начинается с числа 0. Затем головка идет вправо; в первой встретившейся ей при этом пустой ячейке ставится другой разделительный знак ("звездочка"), справа от этого знака записывается число 1 и производится первый шаг вычисления машины *M*, начинающегося с этого числа. Затем головка идет влево до квадратика, и на том участке ленты, где были проделаны два первых шага вычисления *M*, начинающегося с нуля, производится третий шаг этого вычисления. (Для этого нужно, чтобы после первых двух шагов была помечена ячейка, в которой остановилась головка, и метка указывала, на каком этапе вычисление было прервано.) Если на этом шаге головка машины *M* сдвигалась вправо, то производится сдвиг вправо той части записи, которая расположена правее преобразуемого сейчас участка. Далее головка идет вправо, и на том участке ленты, где был выполнен первый шаг вычисления *M*, начинающегося с единицы, производится второй шаг. После этого головка идет дальше вправо; в первой встретившейся ей пустой ячейке ставится еще одна звездочка, за ней записывается число 2 и производится первый шаг начинающегося с него вычисления *M*.

Потом головка снова идет влево до квадратика, после чего производится четвертый шаг вычисления с нулем, третий с единицей, второй с двойкой, затем через звездочку записывается число 3 и над ним производится первый шаг вычисления и т. д.

Таким образом, наша машина параллельно вычисляет (вернее, пытается вычислять) значения функции f для всевозможных значений аргумента n. Этот процесс продолжается до тех пор, пока на каком-то "n-участке" вычисление не остановится. Если эта остановка безрезультатная, так что $f(n)$ не определена, машина продолжает работу с той только разницей, что далее головка проходит этот "n-участок", не изменяя его. Если же вычислено значение $f(n)$, оно сравнивается с x. При $f(n) \neq x$ работа продолжается так же, как в случае безрезультатной остановки; при $f(n) = x$ все содержимое ленты стирается и машина переходит в заключительное состояние.

Ясно, что функция, вычисляемая описанной машиной, определена для тех и только тех x, которые служат значениями функции f, т. е. принадлежат E.

2. Пусть машина M вычисляет функцию f с областью определения E. Рассмотрим машину, входной и выходной алфавиты которой совпадают с входным алфавитом M, работающую так: если в начальной ситуации на ее ленте записано слово x, оно удваивается с поставкой разделительного знака между двумя экземплярами, затем со вторым экземпляром машина работает как M, и если при этом будет получено какое-то значение $f(x)$, то справа от разделительного знака все стирается и он сам тоже, головка становится на граничный маркер и машина переходит в заключительное состояние; если же $f(x)$ не определено, то машина "ломается" или работает вечно. Вычисляемая этой машиной функция определена в точности для тех же x, что и f, и в случае, когда ее значение определено, оно совпадает со значением аргумента; таким образом, множество значений этой функции есть E.

Теорема 4.6. Множество A является рекурсивно перечислимым тогда и только тогда, когда существует разрешимый предикат $R(x, y)$, такой, что $x \in A$ тогда и только тогда, когда $\exists y R(x, y)$.

Теорема 4.7. Всякое разрешимое множество рекурсивно перечислимо.

Доказательство. По машине M, вычисляющей характеристический предикат множества E, легко построить другую машину, которая выдает И, если M выдаст и, и работает вечно, если M выдает Л. Область определения вычисляемой этой машиной функции есть E. Остается воспользоваться теоремой 4.4.

Теорема 4.8 (теорема Поста). Множество тогда и только тогда разрешимо, когда оно само и его дополнение рекурсивно перечислимы.

Доказательство.

1. Пусть множества E и CE (дополнение E) рекурсивно перечислимы. По теореме 4.4 существуют машины Тьюринга M_1 и M_2, вычисляющие функции f_1 и f_2, с областями определения E и CE соответственно. Построим по ним машину (входной алфавит которой будет совпадать с входным алфавитом $A^1_{\text{вх}}$ машины M_1), работающую следующим образом. Если на ее ленте в начальной ситуации записано слово x в алфавите $A^1_{\text{вх}}$, оно удваивается (с постановкой разделительного знака между экземплярами), а затем оба экземпляра обрабатываются параллельно – первый по программе машины M_1, второй по программе машины M_2 (шаги вычислений с первым и вторым экземплярами производятся поочередно – ср. доказательство теоремы 4.4). Поскольку x принадлежит одному и только одному из множеств E и CE, один и только один из этих двух параллельных процессов закончится вычислением значения соответствующей функции. После этого все стирается и записывается и, если вычислено значение f_1, и л, если вычислено значение f_2. Таким образом, наша машина вычисляет характеристический предикат множества E.

2. Обратное утверждение непосредственно вытекает из теорем 4.3 и 4.7.

Обратимся теперь к вопросу о существовании неразрешимых и неперечислимых множеств, а также невычислимых функций. (Разумеется, речь идет о множествах натуральных чисел (или слов в конечных алфавитах) и о функциях, отображающих такие множества в такие же.) Ответить на этот вопрос очень просто, исходя из мощностных соображений. Действительно, мощность множества функций, вычисляемых всевозможными машинами Тьюринга, во всяком случае не превосходит мощности множества кодов всевозможных машин Тьюринга, т. е. слов специального вида в алфавите $\{a_1,...,a_8\}$; но даже множество всех слов в конечном алфавите счетно, в то время как множество всех функций, отображающих подмножества натурального ряда на подмножества натурального ряда, имеет мощность континуума.

Далее, мощность множества всех перечислимых множеств натуральных чисел не превосходит мощности множества вычислимых функций, значения которых пробегают эти множества; таким образом, множество перечислимых множеств натуральных чисел счетно, в то время как множество всех подмножеств натурального ряда имеет мощность континуума. Итак, невычислимые функции и неперечислимые (а значит, и неразрешимые) множества не только существуют, но их "больше", чем вычислимых функций соответственно перечислимых множеств.

Это простое решение страдает, однако, существенным недостатком: оно не дает никакого конкретного примера неперечислимого множества или невычислимой функции. Кроме того, мощностные соображения не позволяют решить столь же естественный вопрос о существовании неразрешимых перечислимых множеств, поскольку множество разрешимых множеств тоже счетно. Если удастся указать пример неразрешимого перечислимого множества, то его дополнение в силу теоремы 4.7 дало бы пример неперечислимого множества, а его характеристический предикат (точнее -- функция, сопоставляющая значениям аргумента истинностные значения значений предиката) – пример невычислимой функции. Прежде чем привести такой пример, введем следующее определение.

Определение 4.7. Стандартный номер кода машины Тьюринга называется геделевским номером, или просто номером, этой машины.

Это определение нуждается, однако, в уточнении: ведь стандартная нумерация слов зависит от выбора алфавита (а также от выбора его пересчета). Коды машин Тьюринга – это цепочки в алфавите $A' = \{a_1, ..., a_8\}$, и мы можем выбрать любой алфавит, содержащий A'. Мы остановимся на алфавите $\{a_1, ..., a_9\}$.

Теорема 4.9. Множество номеров самоприменимых машин Тьюринга рекурсивно перечислимо, но не разрешимо.

Поскольку мы обычно отождествляем слово с его номером, вместо множества номеров мы можем говорить здесь о множестве кодов.

Доказательство. Утверждение о неразрешимости этого множества есть просто другая формулировка теоремы 4.1. Чтобы доказать его перечислимость, рассмотрим машину Тьюринга с входным алфавитом $\{a_1, ..., a_8\}$, работающую следующим образом: если на ее ленте в начальной ситуации записано слово x, она прежде всего, не изменяя этого слова, распознает, является ли оно кодом какой-либо машины Тьюринга; если нет – машина "ломается", если да – удваивает слово x,

вставляя между двумя его экземплярами a_9, и далее работает как универсальная машина, т.е. преобразовывает слово *x* в соответствии с программой той машины, для которой это же слово служит кодом. Если при этом будет получен результат, т.е. если машина с кодом *x* самоприменима, то все содержимое ленты стирается, и машина переходит в заключительное состояние. (В противном случае она "ломается" или работает вечно.) Ясно, что вычисляемая этой машиной функция определена для тех и только для тех слов, которые являются кодами самоприменимых машин Тьюринга. Остается применить теорему 4.5.

Таким образом, два основных способа задания множеств при алгоритмизации оказываются неэквивалентными: перечисление – способ более сильный, чем (эффективное) указание характеристического свойства элементов.

Следствие. Множество номеров несамоприменимых машин Тьюринга неперечислимо.

Доказательство. Пусть *A*, *B* и *C* – соответственно множество номеров несамоприменимых машин, дополнение к множеству номеров самоприменимых машин и множество чисел, не являющихся номерами никаких машин. Множество *C* разрешимо и тем более перечислимо, и если бы *A* также было перечислимо, то ввиду очевидного равенства $B = A \cup C$ по теореме 4.4 оказалось бы перечислимым и *B*, что невозможно в силу теорем 4.8 и 4.9.

С помощью конструкции, использованной в доказательстве теоремы 4.9, мы докажем сейчас еще одну важную теорему. Ее доказательство предоставит нам еще два примера неразрешимых перечислимых множеств (а также способ получать такие примеры в любом числе).

Теорема 4.10. Существуют непересекающиеся рекурсивно перечислимые множества E_1 и E_2 такие, что никакое разрешимое множество не может содержать E_1, не имея при этом общих точек с E_2. (Как иногда говорят, E_1 и E_2 не отделимы разрешимыми множествами.)

Доказательство. Обозначим через E_1 и соответственно через E_2 множество кодов тех машин Тьюринга, которые перерабатывают свой собственный код в пустое и соответственно в непустое слово.

Для доказательства перечислимости этих множеств достаточно рассмотреть машину, использовавшуюся в доказательстве теоремы 4.8.

Пусть в случае E_1 машина переходит в заключительное состояние лишь при условии, что результат, получаемый на этапе, когда машина работает как универсальная, есть пустое слово (пустое слово служит своим собственным кодом), а в случае E_2 – при условии, что этот результат есть непустое слово. Допустим теперь, что существует разрешимое множество E, содержащее E_1 и не пересекающееся с E_2. По машине, вычисляющей характеристический предикат множества E, легко построить другую машину (обозначим ее M), которая каждое число, т. е. каждое слово x в алфавите $\{a_1,...,a_9\}$ перерабатывает в непустое слово, если $x \in E$, и в пустое, если $x \notin E$. Но во что переработает машина M свой собственный код $\kappa(M)$? Если в пустое слово, то тем самым $\kappa(M) \in E_1$, откуда $\kappa(M) \in E$ – а тогда по построению машины она должна перерабатывать $\kappa(M)$ в непустое слово. Если же M перерабатывает $\kappa(M)$ в непустое слово, то $\kappa(M) \in E_2$, откуда $\kappa(M) \notin E$ и, следовательно, M должна перерабатывать $\kappa(M)$ в пустое слово. Полученное противоречие доказывает теорему.

Замечания.

1. Любая неразрешимая алгоритмическая проблема дает пример неразрешимого множества.

2. Существуют и такие множества натуральных чисел, которые неперечислимы вместе со своими дополнениями.

Нельзя ли использовать для перечисления множеств только всюду определенные вычислимые функции? Это лучше отвечало бы интуитивному представлению о перечислении: "запуская" машину последовательно для чисел 0, 1, 2,..., мы получили бы один за другим все элементы множества, никогда не рискуя, что вычисление окажется бесконечным или оборвется безрезультатно. Ответ на этот вопрос, за очевидным исключением пустого множества, оказывается положительным, что вытекает из следующей теоремы.

Теорема 4.11. Всякое непустое рекурсивно перечислимое множество является множеством значений некоторой всюду определенной вычислимой функции.

Доказательство. Пусть машина M вычисляет функцию f с непустым множеством значений E и пусть x_0 – некоторый элемент E. Построим по

машине M новую машину M_1, которая будет работать так же, как машина, построенная в п. 1 доказательства теоремы 4.5, с той только разницей, что зона ленты, где записано данное число x, используется теперь как "счетчик шагов": после имитации каждого шага какого-либо вычисления машины M головка машины M идет в зону счетчика, и содержащееся в этой зоне число уменьшается на единицу. Когда содержимое зоны счетчика оказывается нулем (пустым словом), имитация вычислений машины M прекращается и, если на имитировавшемся перед этим шаге работы машины M как раз было вычислено какое-то значение $f(z)$ функции f, машина записывает это $f(z)$ на ленте (стирая все остальное) и переходит в заключительное состояние; в противном случае она все стирает, записывает на ленте слово x_0 и также переходит в заключительное состояние. Вычисляемая машиной M_1 функция f_1 всюду определена, и легко видеть, что множество ее значений совпадает с E.

Рассмотрим теперь важный подкласс рекурсивно перечислимых множеств, а именно так называемые креативные множества. Это те рекурсивно перечислимые множества, дополнение которых в некотором довольно сильном смысле не является рекурсивно перечислимым. Поэтому мы начнем с рассмотрения класса не рекурсивно перечислимых множеств, которому будут принадлежать эти дополнения.

Пусть A – любое множество, которое не является рекурсивно перечислимым. Тогда если W_x есть какое-нибудь рекурсивно перечислимое множество, содержащееся в A, то должно существовать число $y \in A \setminus W_x$. Существование такого числа говорит о том, что $A \neq W_x$. Оказывается, что для некоторых не рекурсивно перечислимых множеств такое число может быть найдено эффективным образом. Рассмотрим, например, нерекурсивно перечислимое множество $\bar{K} = \{x \mid x \notin W_x\}$. Если $W_x \subseteq \bar{K}$, то не может быть, чтобы $x \in W_x$ (в противном случае $x \in K$ и $W_x \not\subseteq \bar{K}$). Отсюда $x \in \bar{K} \setminus W_x$, и нашим числом, показывающим, что $W_x \neq \bar{K}$, является само число x.

Не рекурсивно перечислимые множества, для которых такое число может быть всегда вычислено таким путем, носят название продуктивных.

√ ***Определение 4.8.*** Множество A называется продуктивным тогда и только тогда, когда существует всюду определенная вычислимая функция g такая, что из $W_x \subseteq A$ следует, что $g(x) \in A \setminus W_x$. Функция g называется продуктивной функцией для множества A (рис. 4.1).

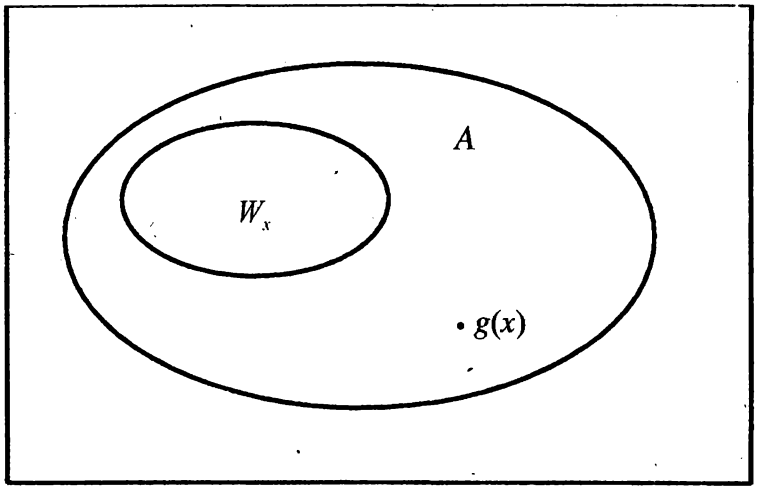


Рис. 4.1. Продуктивное множество

Пример 4.3. Множество $\bar{K}$ продуктивно, причем продуктивная функция есть $g(x) = x$.

Пример 4.4. Следующие множества продуктивны:

$\{x \mid \phi_x \neq 0\}$,

$\{x \mid c \notin W_x\}$ (c – фиксированное число, W_x – область определения ϕ_x);

$\{x \mid c \notin E_x\}$ (c – фиксированное число, E_x – множество значений ϕ_x).

√ ***Определение 4.9.*** Множество A называется креативным, если оно является рекурсивно перечислимым и его дополнение A продуктивно.

Самый простой пример креативного множества дает, конечно, множество K. Креативным множеством является рекурсивно перечислимое множество, которое нерекурсивно в очень сильной степени.

Пример 4.5. Следующие множества являются креативными:

$\{x \mid c \in W_x\}$,

$\{x \mid c \notin E_x\}$,

$A = \{x \mid \phi_x(x) = 0\}$.

Эти множества являются креативными, поскольку их дополнения – продуктивны (см. пример 4.4).

4.3. Примитивно рекурсивные функции

В исходный набор простейших функций естественно включить тривиальные функции вида $0,...,P-1$ $(i=1,...,k)$, а также функции-константы $f(x_1)=m$. Но одних этих функций недостаточно, чтобы получить сколько-нибудь интересный класс функций с помощью подстановки и рекурсии, поэтому мы добавим к ним еще одну, также очень простую и вычислимую тривиальным образом: $S(x)=x+1$. При этом мы сможем вместо всевозможных функций-констант обойтись функцией, тождественно равной нулю: из нее и из $S(x)$ с помощью подстановки получаются все остальные функции-константы.

Функции, получаемые из описанного исходного набора с помощью подстановки и рекурсии, принято называть примитивно рекурсивными (primitive recursive functions, точнее было бы – "простые рекурсивные функции" в противоположность "общим рекурсивным функциям" – general recursive functions).

Всюду в этом подразделе, говоря о функциях, мы будем подразумевать функции, у которых область определения состоит из упорядоченных систем натуральных чисел (для одноместных функций – просто из натуральных чисел) и множество значений также есть подмножество натурального ряда.

Будем называть исходными примитивно рекурсивными функциями функции $e_k^i(x_1,...,x_k)=x_i$, $0(x)=0$, $S(x)=x+1$ (функции e_k^i называют функциями выделения аргумента, или проектирующими функциями).

Рассмотрим, далее, два оператора, действующие на функции: оператор подстановки и оператор рекурсии.

Оператор подстановки применяется к функциям $g_1,...,g_p$ вместимости k и функции h вместимости p ($k, p = 1,2,...$); результат его применения есть функция f вместимости k, определяемая равенством

$$f(x_1,...,x_k) = h\left(g_1(x_1,...x_k),...,g_p(x_1,...x_k)\right) .$$

Оператор рекурсии применяется либо к натуральному числу и функции h вместимости 2, либо к функциям g и h вместимости $n-1$ и $n+1$ соответственно, где $n = 1,2,...$; результат его применения есть функция f вместимости 1 в первом случае и n во втором, удовлетворяющая рекурсивным равенствам. В первом случае это равенства

$$(R_1)\ \begin{cases} f(0) = c \\ f(x+1) = h\left(x, f(x)\right), \end{cases}$$

во втором равенства

$$(R_2)\ \begin{cases} f(x_1,...,x_{n-1},0) = g(x_1,...,x_{n-1}) \\ f(x_1,...,x_{n-1},x_n+1) = h\left(x_1,...,x_{n-1},x_n, f(x_1,...,x_{n-1},x_n)\right). \end{cases}$$

Рекурсивные равенства вполне определяют функцию f: ее значение для $x = 0$ или для $x_n = 0$ однозначно находится из первого равенства, а из второго равенства по значению для очередного x или x_n также однозначно находится значение для $x+1$ и соответственно для x_n+1.

Теперь можно сформулировать основное определение.

Определение 4.10. Примитивно рекурсивной функцией (сокращенно п.р.ф.) называется функция, которая либо принадлежит к числу исходных примитивно рекурсивных функций, либо может быть получена из них с помощью операторов подстановки и рекурсии.

Примитивно рекурсивные функции – это арифметические функции, которые сопоставляются по определенным правилам примитивно рекурсивным описаниям.

Всякая примитивно рекурсивная функция является вычислимой, так как процесс определения такой функции с помощью примитивно

рекурсивного описания дает одновременно и способ вычисления значения функции для любых числовых значений ее аргумента.

Определение 4.11. **Примитивно рекурсивное описание** – это слово вида (k, l, ...), где k,l – натуральные числа и l – число аргументов описания.

Пример 4.6. Функции произвольной вместимости n ($n \succ 1$), тождественно равные нулю: $0_n(x_1,...,x_n) = 0\left(e_n^1(x_1,...,x_n)\right)$.

Пример 4.7. Функции-константы $k(x)$ ($k = 1,2,...$): $1(x) = S\left(0(x)\right)$, $2(x) = S\left(1(x)\right)$ и т. д. Аналогично, с использованием примера 4.3, получаются функции-константы $k_n(x_1,...,x_n) = k$ вместимости $n \succ 1$.

Функции $S_k(x) = x \dot{+} k$ ($k = 2,3,...$): $S_2(x) = S\left(S(x)\right)$, $S_3(x) = S\left(S_2(x)\right)$ и т.д.

Пример 4.8. Функция $x_1 + x_2$ получается с помощью оператора рекурсии из функции $e_1^1(x_1) = x_1$ (в качестве g) и $S\left(e_3^3(x_1,x_2,x_3)\right) = x_3 + 1$ (в качестве h), так что рекурсивные равенства для $x_1 + x_2$ имеют вид

$$\begin{cases} x_1 + 0 = e_1^1(x_1) \\ x_1 + (x_2 + 1) = S\left(e_3^3(x_1,x_2,x_1 + x_2)\right) \end{cases}$$

или

$$\begin{cases} x_1 + 0 = x_1 \\ x_1 + (x_2 + 1) = (x_1 + x_2) + 1 \end{cases}$$

Пример 4.9. Функция $x_1 \cdot x_2$ получается с помощью оператора рекурсии из функций $0(x_1) = 0$ и $e_3^3(x_1,x_2,x_3) + e_3^1(x_1,x_2,x_3) = x_3 + x_1$.

Пример 4.10. Функция $x \dot{-} 1$ ("усеченная разность между x и единицей"), равная $x - 1$ при $x \geq 1$ и нулю при $x \prec 1$, получается с

помощью оператора рекурсии из числа 0 и функции $e_1^1(x_1, x_2) = x_1$, так что рекурсивные равенства имеют вид

$$\begin{cases} 0 \dot{-} 1 = 0 \\ (x+1) \dot{-} 1 = x. \end{cases}$$

В дальнейших примерах, не указывая явно функции g и h, выпишем рекурсивные равенства.

Пример 4.11. Функция $x \dot{-} y$ ("усеченная разность"), равная $x - y$ при $x \geq y$ и нулю при $x \prec y$:

$$\begin{cases} x \dot{-} 0 = x \\ x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1. \end{cases}$$

Пример 4.12. $|x - y| = (x \dot{-} y) + (y \dot{-} x)$.

Пример 4.13. Функция sgx (знак числа x), равная нулю при $x = 0$ и единице при $x \succ 0$:

$$\begin{cases} sg0 = 0 \\ sg(x+1) = 1. \end{cases}$$

Пример 4.14. Функция $\overline{sg}x$, равная единице при $x = 0$ и нулю при $x \succ 0$:

$$\begin{cases} \overline{sg}0 = 1 \\ \overline{sg}(x+1) = 0. \end{cases}$$

Пример 4.15. $\min(x, y) = x \dot{-} (x \dot{-} y)$.

Пример 4.16. $\max(x, y) = y + (x \dot{-} y)$.

Пример 4.17. Функция $r(x, y)$, равная остатку от деления y на x, если $x \neq 0$, и нулю, если $x = 0$:

$$\begin{cases} r(x,0) = 0 \\ r(x, y+1) = \big(r(x,y)+1\big) \cdot sg\big(x \dot{-} \big(r(x,y)+1\big)\big) \end{cases}$$

Пример 4.18. Функция $q(x, y)$, равная целой части частного от деления y на x, если $x \neq 0$, и нулю, если $x = 0$:

$$\begin{cases} q(x,0)=0 \\ q(x,y+1)=q(x,y)+\overline{sg}\left|x-\left(r(x,y)+1\right)\right|. \end{cases}$$

Наряду с примитивно рекурсивными функциями в рассмотрение вводятся примитивно рекурсивные предикаты.

Определение 4.12. Предикат $P(x_1,...x_n)$, определенный на множестве натуральных чисел, называется примитивно рекурсивным, если примитивно рекурсивна функция χ_P, сопоставляющая каждому набору $x_1^0,...,x_n^0$ значений переменных $x_1,...,x_n$ число 1, если предложение $P(x_1^0,...x_n^0)$ истинно, и 0, если это предложение ложно.

Функция χ_P называется характеристической функцией предиката *P*.

Замечание. Точно так же можно было бы определить и предикат, вычислимый по Тьюрингу, как предикат, для которого вычислима по Тьюрингу характеристическая функция.

Приведем примеры примитивно рекурсивных предикатов.

Пример 4.19. $x=c$, где c – произвольная константа: его характеристической функцией служит $\overline{sg}\left|x-c\right|$.

Пример 4.20. $x \succ c$: характеристическая функция $sg(x \dot{-} c)$.

Пример 4.21. $x=y$: характеристическая функция $\overline{sg}\left|x-y\right|$.

Пример 4.22. $x \succ y$: характеристическая функция $sg(x \dot{-} y)$.

Пример 4.23. *x* четно: характеристическая функция задается рекурсивными равенствами

$$\begin{cases} f(0)=1 \\ f(x+1)=\overline{sg}f(x). \end{cases}$$

Пример 4.24. $d(x,y) \equiv$ *x* делится на *y*: характеристическая функция $\overline{sg}r(y,x)\cdot sgy+\overline{sg}x\cdot\overline{sg}y$.

Доказательство примитивной рекурсивности функций и предикатов облегчают следующие теоремы.

Теорема 4.12. Если функция $f(x_1,...x_n)$ $(n=1,2,...)$ примитивно рекурсивна, то таковы же и функции

$$g(x_1,...x_n)=\sum_{i=0}^{x_n} f(x_1,...x_{n-1},i)$$

и

$$h(x_1,...x_n)=\prod_{i=0}^{x_n} f(x_1,...x_n,i).$$

Доказательство. Функция g задается равенствами:

$$\begin{cases} g(x_1,...x_{n-1},0)=f(x_1,...x_{n-1},0) \\ g(x_1,...x_{n-1},x_n+1)=g(x_1,...x_{n-1},x_n)+f(x_1,...x_{n-1},x_n+1). \end{cases}$$

Функция h задается аналогично.

Замечание. Теорема 4.12 останется справедливой, если вместо x_n верхней границей суммирования или перемножения будет произвольная п. р. ф. (достаточно воспользоваться оператором подстановки).

Пример 4.25. Функции $\tau(x)$ и $\sigma(x)$ – число делителей и сумма делителей числа x – примитивно рекурсивны, поскольку:

$$\tau(x)\quad \sum_{i\ 0}^{x} \overline{sg}r(i,x)\cdot 1,$$

$$\sigma(x)\quad \sum_{i\ 0}^{x} i\ \overline{sg}r(i,x)$$

(здесь условно считается $\tau(0)=\sigma(0)=0$).

Теорема 4.13. Если предикаты $P(x_1,...,x_n)$ и $Q(x_1,...,x_n)$ примитивно рекурсивны, то таковы же и предикаты $\neg P(x_1,...,x_n)$, $P(x_1,...,x_n)\ \ Q(x_1,...,x_n)$, $P(x_1,...,x_n)\ \ Q(x_1,...,x_n)$, $P(x_1,...,x_n)\ \ Q(x_1,...,x_n)$

Доказательство. Достаточно доказать утверждение теоремы для $\neg P$ и $P \wedge Q$. Если $\varphi(x_1,...,x_n)$ и $\chi(x_1,...,x_n)$ – характеристические функции предикатов P и Q соответственно, то для $\neg P$ и $P \wedge Q$ характеристическими функциями будут $\overline{sg}\varphi(x_1,...,x_n)$ и $\varphi(x_1,...,x_n)\cdot\chi(x_1,...,x_n)$.

Пример 4.26. Из теоремы 4.12 и примеров 4.19–4.23 вытекает, что примитивно рекурсивны предикаты $x \neq y$, $x \leq y$, $x \prec y$, $a \prec x \prec b$ (a, b -- константы), M есть общий делитель чисел x и y и т. п.

Теперь рассмотрим еще один оператор, применяемый к предикатам и переводящий их в функции – так называемый оператор минимизации. Результатом его применения к предикату $P(x_1,...x_n,y)$ является функция вместимости n, сопоставляющая каждому набору $x_1^0,...,x_n^0$ значений переменных $x_1,...x_n$ наименьшее значение y^0 переменной y, при котором предложение $P(x_1^0,...,x_n^0,y^0)$ истинно (если такое значение существует). Этот оператор не сохраняет примитивную рекурсивность, а результатом его применения может быть не всюду определенная функция. Но в ряде важных случаев его можно заменить оператором ограниченной минимизации, который применяется к предикату $P(x_1,...x_n,y)$ и функции $\varphi(x_1,...x_n)$ ($n=1,2,...$). Результат его применения есть функция $f(x_1,...x_n)$, значение которой для каждого набора $x_1^0,...,x_n^0$ значений переменных $x_1,...x_n$ определяется следующим образом: если хотя бы для одного числа y, не превосходящего $\varphi(x_1^0,...,x_n^0)$, предложение $P(x_1^0,...,x_n^0,y^0)$ истинно, то $f(x_1^0,...,x_n^0)$ равно наименьшему из таких y; в противном случае $f(x_1^0,...,x_n^0)=0$. Эту функцию $f(x_1,...x_n)$ принято обозначать $\underset{y\leq\varphi}{\mu y} P(x_1,...,x_n,y)$. Оператор ограниченной минимизации сохраняет примитивную рекурсивность, как показывает следующая теорема.

Теорема 4.14 (об ограниченной минимизации). Если предикат $P(x_1,...x_n,y)$ и функция $\varphi(x_1,...x_n)$ примитивно рекурсивны, то функция $\underset{y\le\varphi}{\mu y} P(x_1,...,x_n,y)$ также рекурсивна.

Доказательство. Если $\chi(x_1,...,x_n,y)$ -- характеристическая функция предиката *P*, то $\prod_{i=0}^{k} \overline{sg}\,\chi(x_1,...,x_n,i)=1$ для всех *k*, меньших наименьшего *y*, для которого $P(x_1,...,x_n,y)$ истинно, и 0 для всех *k*, больших или равных этому *y*; второй сомножитель равен единице, если $P(x_1,...,x_n,y)$ истинно хотя бы для одного $y \le \chi(x_1,...,x_n)$, и нулю в противном случае.

$$\underset{y\le\varphi}{\mu y} P(x_1,...,x_n,y)=\left[\sum_{k=0}^{\varphi}\prod_{i=0}^{k}\overline{sg}\,\chi(x_1,...,x_n,i)\right]\cdot sg\sum_{m=0}^{\varphi}\chi(x_1,...,x_n,m).$$

Пример 4.27. Функция (x,y) – наибольший общий делитель чисел *x* и *y*:

$$(x,y)=\underset{z\le\min(x,y)}{\mu z}\ D(x,y,z).$$

Пример 4.28. Функция p_x – простое число, имеющее номер *x* в последовательности простых чисел, расположенных в порядке возрастания (причем считается $2=p_0$):

$$\begin{cases} p_0=2 \\ p_{x+1}=\underset{y\le p_x!+1}{\mu y}\ (y\succ p_x\cap p(y)). \end{cases}$$

Таким образом, здесь во втором рекурсивном равенстве $h(x_1,x_2)=\underset{y\le x_2!+1}{\mu y}\ (y\succ x_2\cap p(y))$. При выборе ограничивающей функции использовано евклидово доказательство существования сколь угодно больших простых чисел: каково бы ни было $N=2,3,...$, среди $N+1$,

$N+2,\ldots N!+1$ должно быть хоть одно простое, поскольку $N!+1$ делится на какое-то простое число, но ни на одно из чисел 2,..., *N* делиться не может.

Как было отмечено ранее, все примитивно рекурсивные функции вычислимы в интуитивном смысле; запас этих функций, как видно из теорем и приведенных примеров, весьма широк. Но все же не каждая вычислимая функция примитивно рекурсивна. Далее мы построим пример функции, допускающей вычисление с помощью довольно очевидной, хотя и громоздкой процедуры и в то же время не примитивно рекурсивной. Для этого введем новое понятие.

Будем сопоставлять примитивно рекурсивным функциям натуральные числа, которые будем называть их геделевскими номерами или просто номерами, следующим образом:

а) число 2 будет номером функции $0(x)$;

б) число 3 будет номером функции $S(x)$;

в) функции e_k^i будут нумероваться последовательными простыми числами p_2, p_3, ... таким образом, что с возрастанием *k* номера возрастают, а при одинаковых *k* функция с большим *i* имеет больший номер. Например, e_1^1 имеет номер 5, e_2^1 – 7, e_2^2 – 11 и т. д.;

г) если одноместная функция *f* получается с помощью оператора рекурсии из числа *c* и двухместной функции *h*, имеющей номер *j*, то число $2^{c+1}\cdot 5^j$ будет номером функции *f*;

д) если $n>1$ и *n*-местная функция *f* получается с помощью оператора рекурсии из $(n-1)$-местной функции g и $(n+1)$-местной функции *h*, имеющих номера *i* и *j* соответственно, то число $3^i\cdot 5^j$ будет номером функции *f*;

е) если $f(x_1,\ldots x_k)=h\left(g_1(x_1,\ldots x_k),\ldots,g_m(x_1,\ldots x_k)\right)$ и функции *h*, $g_1,\ldots,g_m$ имеют номера *i*, $j_1,\ldots,j_m$ соответственно, то число $p_3^i p_4^{j_1}\ldots p_{m+3}^{j_m}$ будет номером функции *f*.

Поскольку каждую п. р. ф. можно получить из исходных бесконечно многими способами, всякая п. р. ф. имеет бесконечно много номеров.

Нетрудно указать эффективную процедуру, позволяющую по любому

натуральному числу распознать, является ли оно номером какой-либо п. р. ф., и в случае положительного ответа указать способ ее вычисления. Эта процедура называется расшифровкой натурального числа; производится она шаг за шагом: сначала расшифровывается нуль, потом единица, потом двойка и т. д. Расшифровка нуля состоит просто в констатации, что нуль не является номером никакой п. р. ф. Пусть $n > 0$ и все числа $0, \ldots, n-1$ уже расшифрованы. Тогда расшифровка числа n состоит в следующем. Прежде всего проверяем, является ли n простым числом. Если да – находим исходную п. р. ф. с номером n (это не представляет труда, и способ вычисления такой функции очевиден), и на этом расшифровка заканчивается. Если нет – проверяем, имеет ли n вид $2^{c+1} \cdot 5^j$, и если имеет – является ли j номером какой-либо двухместной п. р. ф. (Число j, поскольку оно меньше n, уже расшифровано, а так как при расшифровке номера функции указывается способ ее вычисления, то должна быть, разумеется, указана и ее вместимость.) Если j есть номер двухместной п. р. ф. h, то n является номером одноместной п. р. ф., получающейся из c и h с помощью оператора рекурсии, и из способа вычисления h получается способ вычисления f, так что расшифровка на этом заканчивается. Если j не есть номер двухместной п. р. ф., мы констатируем, что n не есть номер п. р. ф., и расшифровка заканчивается. Если же n не имеет вида $2^{c+1} \cdot 5^j$, мы проверяем, имеет ли n вид $3^i \cdot 5^j$ и если да – являются ли i и j номерами каких-либо п. р. ф. g и h вместимости $k-1$ и $k+1$ соответственно для некоторого $k = 2, 3, \ldots$. В случае положительного ответа n есть номер k-местной п. р. ф. f, получающейся из g и h с помощью оператора рекурсии, и способы вычисления g и h дают способ вычисления f. В противном случае проверяем, имеет ли n вид $p_3^i p_4^{j_1} \ldots p_{m+3}^{j_m}$ для каких-либо целых положительных $m, i, j_1, \ldots j_m$, и если да – верно ли, что i является номером некоторой m-местной п. р. ф. h, а $j_1, \ldots j_m$ – номерами некоторых п. р. ф. $g_1, \ldots, g_m$ одной и той же вместимости k. В случае положительного ответа n есть номер п. р. ф. $h\big(g_1(x_1, \ldots, x_k), \ldots, g_m(x_1, \ldots, x_k)\big)$, способ вычисления которой получается из способов вычисления h, $g_1, \ldots, g_m$, в противном случае остается констатировать, что n не является номером никакой п. р. ф.

Обозначим теперь через $\psi(x,y)$ двухместную функцию, определяемую следующим образом: если x есть номер некоторой одноместной п. р. ф. χ, то $\forall y\left(\psi(x,y)=\chi(y)\right)$; в противном случае $\forall y\left(\psi(x,y)=0\right)$. Поскольку мы располагаем эффективной процедурой, позволяющей для любого числа x узнать, является ли оно номером одноместной п. р. ф., и в случае положительного ответа указать способ вычисления этой п. р. ф., функция ψ должна быть признана в интуитивном смысле вычислимой (разумеется, она вычислима и по Тьюрингу). В то же время можно показать, что она не примитивно рекурсивна. В самом деле, в противном случае вместе с ней была бы примитивно рекурсивна также и одноместная функция $\psi(x,x)+1$ (поскольку $\psi(x,x)=\psi\left(e_1^1(x),e_1^1(x)\right)$). Поэтому она имела бы в нашей нумерации некоторый номер x_0 (у нее было бы даже бесконечно много номеров, но нам достаточно одного). А тогда по определению функции ψ оказалось бы $\psi(x_0,y)=\psi(y,y)+1$ для любого y, и, в частности, $\psi(x_0,x_0)=\psi(x_0,x_0)+1$. Полученное противоречие доказывает, что ψ не есть п. р. ф.

Итак, мы убедились, что не все вычислимые функции примитивно рекурсивны. Тем не менее примитивная рекурсивность – понятие очень естественное и полезное. Примитивно рекурсивных функций более чем достаточно для всех практических вычислительных задач, да и в теоретических математических исследованиях вне математической логики и смежных с ней областей вряд ли можно столкнуться с конкретной числовой функцией, которая не была бы примитивно рекурсивной.

В то же время класс примитивно рекурсивных функций можно естественным образом расширить так, чтобы полученный класс совпал с классом всех вычислимых функций.

4.4. Теорема о существовании универсальной частично рекурсивной функции

Чтобы получить все вычислимые функции тем же способом, которым в подразд. 4.3 были получены п. р. ф., достаточно добавить к операторам

подстановки и рекурсии упоминавшийся выше оператор минимизации. Однако нам придется применять этот оператор и к таким предикатам, которые определены не для всех, а только для некоторых наборов натуральных чисел. Функции, которыми мы сейчас займемся, тоже не всегда будут всюду определенными. Но все функции и предикаты, рассматриваемые в настоящем подразделе, определены на множествах, состоящих из упорядоченных наборов натуральных чисел, и значения функций – также натуральные числа; в дальнейшем это специально оговариваться не будет.

В применении к всюду определенным предикатам оператор минимизации в том виде, как он упоминался выше, сохраняет вычислимость (хотя и не сохраняет свойство быть всюду определенным). В самом деле, если $P(x_1,...,x_n,y)$ – всюду определенный вычислимый предикат, то для всякого набора $x_1^0,...,x_n^0$ значений $x_1,...,x_n$ мы можем, последовательно вычисляя $P(x_1^0,...,x_n^0,0)$, $P(x_1^0,...,x_n^0,1)$ и т. д., найти наименьшее число y^0, для которого $P(x_1^0,...,x_n^0,y^0)$ И, если только оно существует (в противном случае этот процесс будет продолжаться вечно). Но для не всюду определенных предикатов такой способ не годится: если, например, $P(x_1^0,...,x_n^0,0)$ не определено, а $P(x_1^0,...,x_n^0,1)$ И, то, пытаясь вычислить $P(x_1^0,...,x_n^0,0)$ (например, с помощью машины Тьюринга), мы, быть может, будем вынуждены работать вечно и так никогда и не узнаем, что это значение не существует, а следующее есть как раз то, которое нам нужно. Поэтому для общего случая оператор минимизации определяется так: он применяется к предикату P вместимости $k+1$ ($k = 1, 2, ...$), и результат его применения есть функция f вместимости k, такая, что для всякого упорядоченного набора $x_1^0,...,x_n^0$, для которого существуют такие числа y, что $P(x_1^0,...,x_n^0,y)$ И и при этом, если $y \succ 0$, для всех $i = 0,...,y-1$ значения $P(x_1^0,...,x_n^0,i)$ определены, значение $f(x_1^0,...,x_k^0)$ равно наименьшему из таких y; если же таких y не существует, то $f(x_1^0,...,x_k^0)$ не определено.

Результат применения к предикату P оператора минимизации принято обозначать $\mu y P(x_1,...,x_k,y)$.

Мы будем теперь называть функцию *частично рекурсивной* (сокращенно ч. р. ф.), если она либо принадлежит к числу исходных п. р. ф., либо может быть получена из них с помощью операторов подстановки, рекурсии и минимизации.

Всюду определенная частично рекурсивная функция называется *общерекурсивной* (сокращенно о. р. ф.).

Идея использования рекурсии для решения задач, связанных с основаниями математики, восходит к Т. Сколему и Д. Гильберту. Точное понятие примитивно рекурсивной функции было введено К. Геделем; он называл эти функции просто рекурсивными. Термин "примитивная рекурсия" предложила в 1934 г. Р. Петер для обозначения рекурсии простейшего вида (описанной в начале предыдущего подраздела) в отличие от более сложных рекурсий, сводящихся к примитивной. Общерекурсивные и частично рекурсивные функции были введены Ж. Эрбраном и К. Геделем; наиболее распространенную сейчас форму их определения – с помощью оператора минимизации, использованную здесь, предложил С. К. Клини. Он же ввел термины primitive recursive function, general recursive function, partial recursive function.

Понятие частично рекурсивной функции совпадает по объему с понятием функции, вычислимой по Тьюрингу. Это доказывается с помощью двух теорем.

Теорема 4.15. Если V – конечный алфавит мощности > 1 и η – стандартная нумерация слов в этом алфавите, то для всякой ч. р. ф. $f(x_1,...,x_n)$ можно построить машину Тьюринга с входным и выходным алфавитами, совпадающими с V, которая будет вычислять функцию $\eta^{-1} f(\eta v_1,...,\eta v_n)$.

Теорема 4.16. Если V – конечный алфавит мощности > 1, η – стандартная нумерация слов в этом алфавите и $f(v_1,...,v_n)$ – функция, вычисляемая некоторой машиной Тьюринга с входным и выходным алфавитами, совпадающими с V, то функция $\eta f(\eta^{-1}x_1,...,\eta^{-1}x_n)$ частично рекурсивна.

Идея доказательства теоремы 4.15 состоит в том, что достаточно построить машины для вычисления исходных п. р. ф. и для каждого из трех операторов указать способ, позволяющий по машинам,

вычисляющим функции, к которым этот оператор применяется, построить машину для вычисления функции, получающейся в результате его применения.

Для доказательства теоремы 4.16 используется метод арифметизации конструктивных процессов. Арифметизация состоит в том, что конструктивные процедуры некоторого строго определенного класса нумеруются натуральными числами так, что всякую процедуру можно однозначно "расшифровать" по ее номеру. Свойства процедур превращаются тогда в свойства чисел, преобразования процедур – в числовые функции. Этот метод, изобретенный К. Геделем и впервые примененный им в доказательстве знаменитой теоремы о неполноте арифметики, настолько поразителен как по простоте и изяществу идеи, так и по силе получаемых с его помощью результатов, что напрашивается сравнение с другой арифметизацией, по праву причисляемой к величайшим достижениям математики – декартовой арифметизацией геометрии.

Для доказательства частичной рекурсивности вычислимых функций метод арифметизации применил С.К. Клини.

Теорема 4.17 (теорема Клини о нормальной форме). Существуют такие примитивно рекурсивные предикаты T_1, T_2,... вместимости 3, 4,... соответственно и такая одноместная п. р. ф. P, что для всякого $n = 1, 2, \ldots$ любая n-местная ч. р. ф. f может быть представлена в виде $f(x_1, \ldots, x_n) = P\left(\mu z T_n(i_0, z, x_1, \ldots, x_n)\right)$ при некотором подходящем i_0.

Из этой теоремы следует, что при получении произвольной частично рекурсивной функции из исходных всегда можно обойтись одним применением оператора минимизации.

Другое очевидное следствие из теоремы 4.17 – существование при любом $n = 1, 2, \ldots$ $(n+1)$-местной ч. р. ф., универсальной для n-местных ч. р. ф., т. е. такой ч. р. ф. $\Theta(y_0, x_1, \ldots, x_n)$, что для всякой n-местной ч. р. ф. χ при подходящем y_0 справедливо тождество $\chi(x_1, \ldots, x_n) \equiv \Theta(y_0, x_1, \ldots, x_n)$. Действительно, этим свойством обладает функция $P\left(\mu z T_n(y, z, x_1, \ldots, x_n)\right)$.

Замечания.

1. Построенная в подразд. 4.4 двухместная вычислимая функция ψ является универсальной для одноместных п. р. ф.

При доказательстве того, что сама ψ не есть п. р. ф., использовалась только ее универсальность; таким образом, фактически мы доказали, что никакая функция, универсальная для одноместных п. р. ф., не может сама быть примитивно рекурсивной. Легко обобщить это доказательство на случай функций любой вместимости. Точно так же функция, универсальная для n-местных о. р. ф., не может быть общерекурсивной – это можно доказать тем же самым рассуждением.

2. Еще одно следствие теоремы 4.17 – несохранение примитивной рекурсивности оператором минимизации.

Теорема 4.18. Класс функций М, вычислимых по Маркову, совпадает с классом функций Т, вычислимых по Тьюрингу, и, следовательно, с классом частично рекурсивных функций Ч.

Доказательство совпадения классов М и Ч проводится по той же схеме, что и доказательство совпадения классов Т и Ч.

4.5. Формальная арифметика

Наряду с чрезвычайно общими понятиями современная математика продолжает пользоваться более старыми и более конкретными – такими, как число, точка, прямая, плоскость, к которым в конечном счете восходят и все более абстрактные понятия. Возникает вопрос, в какой мере эти конкретные понятия можно формализовать средствами математической логики. А так как все эти, да и все остальные математические концепции так или иначе опираются на понятие натурального числа, первым шагом на пути решения этого вопроса должна быть попытка формализовать арифметику натуральных чисел. Такая попытка действительно была предпринята (в основном по инициативе Д. Гильберта) в первой трети XX в. и привела к совершенно неожиданным результатам.

Формализация арифметики начинается с определения формального логического языка L, адекватного для формулировки на нем утверждений обычной арифметики натуральных чисел. Язык L имеет свой собственный алфавит, который состоит из символов 0, 1, +, ×, = (содержательный смысл которых очевиден) и следующих символов, обозначающих логические операции: $\neg$ ("не"), $\wedge$ ("и"), $\vee$ ("или"), $\Rightarrow$ ("следует"), $\forall$ ("для всех"), $\exists$ ("существует"). Здесь мы будем пользоваться символами $\forall$, $\exists$ только в рамках языка L, а слова "для всех" и "существует" будем применять, только проводя неформальные рассуждения. Кроме того, в языке $\exists$ имеются символы x, y, M, ..., которые обозначают переменные, а также скобки вида (и).

Утверждениями (или формулами) языка L называются осмысленные

конечные последовательности символов из алфавита L. Например, утверждение

$$\exists y\left(y\times(1+1)=x\right)$$

есть формальная запись неформального утверждения "x четно". Удобно писать сокращенно вместо 1+1 символ 2, вместо (1+1)+1 –символ 3 и т. д. для всех натуральных чисел. Тогда ложное неформальное утверждение "число 5 четно" формально выражается на языке L с помощью утверждения

$$\exists y\left(y\times 2=5\right).$$

Подобным же образом на языке L выражаются и многие другие неформальные утверждения обычной арифметики. Скажем, утверждение "$x\succ y$" принимает вид

$$\exists z\left(\neg(z=0)\wedge(y+z=x)\right).$$

(утверждение $\neg(z=0)$ часто сокращенно записывают в виде $z\neq 0$).

Вместо "x – простое число" можно написать

$$(x\neq 0)\wedge(x\neq 1)\wedge\forall y\forall z\left(x=y\times z\Rightarrow(y=1\vee z=1)\right).$$

Обозначим через $\mathfrak{I}$ множество всевозможных осмысленных утверждений языка L. Тогда $\mathfrak{I}$ распадается на два важных подмножества, а именно:

$\mathcal{T}$ – множество всех утверждений, истинных в обычной арифметике натуральных чисел;

$\mathcal{F}$ – множество всех утверждений, ложных в обычной арифметике натуральных чисел.

Естественное желание математика – узнать как можно больше о свойствах множества $\mathcal{T}$. Например, с точки зрения теории вычислимости важен вопрос: будет ли множество $\mathcal{T}$ рекурсивным или даже рекурсивно перечислимым?

Другой вопрос, существенный как с математической, так и с философской точки зрения, состоит в следующем: существует ли какое-нибудь простое подмножество множества $\mathcal{T}$ (система аксиом), из которого выводятся все остальные утверждения $\mathcal{T}$?

Как мы скоро увидим, ответ на оба этих вопроса оказывается отрицательным.

Первому вопросу можно дать точную формулировку с помощью

стандартной процедуры кодировки. Используя процедуру нумерации, можно указать способ эффективного перечисления (без повторений) множества $\mathfrak{I}$. Допустим, что это уже проделано, и обозначим через θ_n (n + 1)-е утверждение из $\mathfrak{I}$ в этом перечислении, так что

$$\mathfrak{I} = \{\theta_0, \theta_1, \theta_2, ...\}.$$

Эффективность этого перечисления означает, что для каждого заданного n можно эффективно найти и записать утверждение θ_n, и, обратно, для любого утверждения σ из $\mathfrak{I}$ можно эффективно вычислить кодирующее число n такое, что $\sigma = \theta_n$.

Такая кодировка ставит в соответствие каждому множеству утверждений $\aleph$ множество натуральных чисел $X = \{n \mid \theta_n \in \aleph\}$.

Мы будем говорить, что $\aleph$ рекурсивно, если X рекурсивно; $\aleph$ рекурсивно перечислимо, если X рекурсивно перечислимо и т.д. Тем самым первому вопросу придается точный смысл.

Одним из ключевых результатов, который делает теорию вычислимости крайне полезным средством исследования формальной арифметики, является следующая, принадлежащая Геделю, лемма, приводимая без доказательства.

Лемма 4.1. Пусть $P(x_1, ..., x_n)$ -- любой разрешимый предикат. Тогда можно построить утверждение $\sigma(x_1, ...x_n)$ языка L, которое будет являться формальным аналогом предиката $P(x_1, ..., x_n)$ в следующем смысле: для любых $a_1, ..., a_n \in \square$ $P(a_1, ..., a_n)$ истинно тогда и только тогда, когда $\sigma(a_1, ...a_n) \in \mathcal{T}$.

Рассмотрим теперь креативное множество K. Тогда из леммы 4.1 немедленно вытекает следующий результат.

Лемма 4.2. Для любого $n \in \square$:

$n \in K$ тогда и только тогда, когда $n \in K \in \mathcal{T}$;

$n \notin K$ тогда и только тогда, когда $n \notin K \in \mathcal{T}$.

Теперь, чтобы ответить на первый вопрос, нам понадобится лишь следующая лемма.

Лемма 4.3. Существует всюду вычислимая функция g такая, что для любого n утверждение $\theta_{g(n)}$ есть $n \notin K$.

Доказательство следует из эффективности процедуры кодировки утверждений, поскольку для любого данного n мы можем эффективно выписать утверждение

$$n \notin K \left(= \neg \exists y \sigma_R(n, y)\right).$$

Перейдем теперь к ответу на второй вопрос.

Теорема 4.19. Множество $\mathcal{T}$ не является рекурсивно перечислимым; в действительности множество $\mathcal{T}$ продуктивно.

Доказательство. Пусть $\mathbb{T} = \{n \mid \theta_n \in \mathcal{T}\}$. Беря функцию g согласно лемме 4.3, мы имеем:

$n \in \bar{K} \Leftrightarrow n \notin K$,

$n \in \bar{K} \Leftrightarrow n \notin K \in \mathcal{T}$ (по лемме 4.2),

$n \in \bar{K} \Leftrightarrow g(n) \in \mathbb{T}$ (по лемме 4.3).

Таким образом, поскольку множество $\bar{K}$ не является рекурсивно перечислимым, рекурсивно перечислимым не может быть и множество $\mathbb{T}$.

4.6. Теорема Геделя о неполноте математики

Упрощенный вариант теоремы неполноты Геделя легко следует из теоремы 4.19. Опишем сначала постановку этой знаменитой задачи.

Рассмотрим второй вопрос, сформулированный в подразд. 4.5. Ему можно придать точный смысл с помощью понятия формальной системы. Формальная система $(\mathcal{A}, \mathcal{D})$ (для данного языка L) состоит из множества $\mathcal{A} \subseteq \Im$ (системы аксиом) и явного определения $\mathcal{D}$ понятия формального доказательства или вывода утверждения из $\Im$ из этих аксиом. При этом должны быть выполнены следующие условия:

4.1, а) доказательства являются конечными объектами (и, следовательно, их можно кодировать);

4.1, б) явное определение $\mathcal{D}$ понятия доказательства должно быть таким, что если множество $\mathcal{A}$ рекурсивно, то отношение "p есть доказательство утверждения σ из аксиом $\mathcal{A}$" разрешимо.

Теперь мы можем следующим образом сформулировать в точном виде второй вопрос: существует ли формальная система в L такая, что:

4.2, а) множество $\mathcal{A}$ рекурсивно (так что простота набора аксиом понимается в довольно широком смысле);

4.2, б) множество доказуемых утверждений в точности совпадает с $\mathcal{T}$?

Условие 4.2, б) ставит проблему перед философом, который хотел бы определить само понятие арифметической истины с помощью формальной системы. Для него такое условие не имеет смысла, и его следовало бы заменить какими-нибудь другими условиями, отражающими те свойства, которым должна удовлетворять истина, например:

4.2, б′) непротиворечивость: не существует утверждения σ такого, что доказуемы оба утверждения σ и $\neg\sigma$;

4.2, б′′) полнота: для любого утверждения σ одно из утверждений σ и $\neg\sigma$ обязательно доказуемо.

Теорема Геделя в ее упрощенном варианте говорит о том, что не существует формальной системы арифметики, удовлетворяющей условиям 4.2, а и б. Этот факт выводится из теоремы 4.19 и приводится ниже. Полный вариант теоремы Геделя [1931], усовершенствованный Россером, показывает, что не существует формальной системы арифметики, удовлетворяющей условиям 4.2, а, б′, б′′. Другими словами, любая непротиворечивая формальная система арифметики с рекурсивной системой аксиом неполна.

Теорема Геделя о неполноте арифметики поколебала оптимистические надежды Гильберта на полное решение вопросов оснований математики с помощью аксиоматического метода.

Для доказательства упрощенной теоремы Геделя нам понадобится следующая лемма.

Лемма 4.4. В любой рекурсивно аксиоматизированной формальной системе множество доказуемых утверждений рекурсивно перечислимо.

Доказательство. Пусть $\mathcal{P}r$ --набор доказуемых утверждений из $\mathfrak{I}$. Поскольку все доказательства конечны, их можно эффективно перечислить. Тогда из того, что множество аксиом $\mathcal{A}$ рекурсивно, следует, что предикат $M(x,y) \equiv$ "y есть номер доказательства утверждения θ_x из аксиом $\mathcal{A}$" разрешим в силу условия 4.1, б. Тогда θ_x доказуемо $\Leftrightarrow$ существует y такое, что $M(x,y)$ истинно.

Отсюда по теореме 4.7 множество $\mathcal{P}r$ является рекурсивно перечислимым. Сформулируем теперь теорему.

Теорема 4.20 (упрощенная теорема Геделя о неполноте). Пусть $(\mathcal{A}, \mathcal{D})$ – рекурсивно аксиоматизированная формальная система, в которой все доказуемые утверждения истинны. Тогда существует утверждение σ, которое является истинным, но недоказуемым (и, следовательно, утверждение $\neg\sigma$ также недоказуемо).

Доказательство. По лемме 4.4 множество $\mathcal{P}r$ доказуемых утверждений будет рекурсивно перечислимым, причем нам дано, что $\mathcal{P}r \subseteq \mathcal{T}$. Далее, по теореме 4.19 множество $\mathcal{T}$ не является рекурсивно перечислимым, так что мы немедленно получаем утверждение $\sigma \in \mathcal{T} \setminus \mathcal{P}r$, которое и будет истинным, но недоказуемым. Ясно, что утверждение $\neg\sigma$ также не является доказуемым (в противном случае $\neg\sigma$ было бы истинно). [Пользуясь продуктивностью множества $\mathcal{T}$, утверждение теоремы можно усилить и сказать, что утверждение σ может быть эффективно получено из определения формальной системы (которое дает индекс множества $\mathcal{P}r$)].

Чтобы лучше понять доказательство полной теоремы Геделя, которое будет приведено далее, полезно рассмотреть внутренний механизм получения утверждения σ с помощью теоремы 4.19 в только что изложенном доказательстве.

Будем говорить, что некоторое утверждение опровержимо, если его отрицание доказуемо. Рассмотрим множества натуральных чисел Pr* и Ref*, определяемые следующим образом:

$$\mathrm{Pr}^* = \{n \mid n \in K \text{ доказуемо}\},$$

$$\mathrm{Ref}^* = \{n \mid n \in K \text{ опровержимо}\},$$

$$= \{n \mid n \notin K \text{ доказуемо}\},$$

$$= \{n \mid \theta_{g(n)} \in \mathcal{P}r$$

(где g есть вычислимая функция, даваемая леммой 4.3, которую мы использовали при доказательстве теоремы 4.19). Предположение о том, что доказуемые утверждения истинны, означает, в частности, что $\mathrm{Pr}^* \subseteq K$ и $\mathrm{Ref}^* \subseteq \overline{K}$. Далее, множество Ref* является рекурсивно перечислимым (это следует из того факта, что $n \in \mathrm{Ref}^* \Leftrightarrow \theta_{g(n)} \in \mathcal{P}r$ и что множество

$\mathcal{P}r$ является рекурсивно перечислимым); поэтому существует число m такое, что $\mathrm{Ref}^* = W_m$.

В силу продуктивности множества $\bar{K}$ мы немедленно получаем, что $m \in \bar{K} \setminus \mathrm{Ref}^*$, т.е. что $m \notin K$ и, следовательно, что утверждение $m \notin K$ недоказуемо. Обозначая утверждение $m \notin K$ через σ, мы видим, что σ истинно, но недоказуемо (как и выше, легко видеть, что утверждение $\neg\sigma$ тоже недоказуемо). Это рассуждение проиллюстрировано на рис. 4.2.

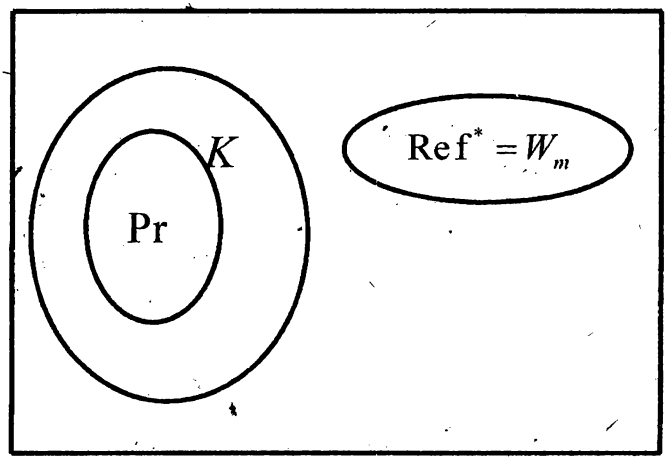


Рис. 4.2. Упрощенный вариант теоремы Геделя о неполноте (теорема 4.20)

Для сравнения с доказательством полной теоремы Геделя, которое будет приведено ниже, заметим, что недоказуемость $\neg\sigma$ можно рассматривать как следствие того факта, что $\mathrm{Pr}^* \subseteq K$: а именно, $m \notin K$ (как и выше), так что $m \notin \mathrm{Pr}^*$, т.е. утверждение $m \notin K$ недоказуемо. Тогда, применяя правила формального вывода, получаем, что утверждение $m \quad K$ (т.е. $m \notin W_m$) тоже недоказуемо.

Таким образом, становится понятным содержательный смысл утверждения σ: σ есть формальный эквивалент утверждения $m \notin K$, т.е. $m \notin W_m$. В то же время

$$m \notin W_m \Leftrightarrow m \notin \mathrm{Ref}^*$$

$$m \notin W_m \Leftrightarrow m \notin \mathrm{K} \text{ недоказуемо}$$

$$m \quad W_m \quad \sigma \quad \mathcal{P}r$$

Тем самым σ есть формальный аналог утверждения "σ недоказуемо", т. е., нестрого говоря, утверждение σ утверждает "я не являюсь доказуемым". Это напоминает парадокс лжеца, касающийся следующего неформального утверждения:

$$\Lambda \equiv \text{“я лгу”}.$$

Неформальный анализ утверждения Λ приводит к следующему парадоксальному выводу: Λ истинно тогда и только тогда, когда Λ неистинно.

Однако если те же самые неформальные рассуждения применить к неформальному утверждению "я не являюсь доказуемым", то парадокса можно избежать, если считать, что доказуемость и истинность – это разные вещи. Именно к такому неформальному заключению приводит нас доказательство теоремы 4.20.

Покажем теперь, как можно усовершенствовать идею доказательства теоремы 4.20 с тем, чтобы избежать любого обращения к понятию истинности.

Фиксируем на время одну формальную систему арифметики, называемую арифметикой Пеано. Аксиомы этой системы образуют рекурсивное подмножество $\mathcal{T}$, которое называется аксиомами Пеано (см. файл материалов). Эти аксиомы описывают простые свойства операции взятия следующего элемента □ и рекурсивное определение сложения и умножения в терминах этой операции, а также включают систему аксиом, описывающих принцип индукции на □. Понятие формального доказательства берется из исчисления предикатов первого порядка.

Для наших целей нам необходимо знать следующий важный факт арифметики Пеано, которому посвящена существенная часть доказательства Геделя.

Лемма 4.5. Пусть $P(x_1,...,x_n)$ – разрешимый предикат, и пусть $\sigma(x_1,...,x_n)$ – утверждение в L, являющееся формальным эквивалентом предиката $P(x_1,...,x_n)$ в смысле леммы 4.1. Тогда предикат P представим

в арифметике Пеано в следующем смысле. Для любых $a_1,...a_n \in \square$:

а) если $P(a_1,...,a_n)$ истинно, то $\sigma(a_1,...,a_n)$ доказуемо;

б) если $P(a_1,...,a_n)$ ложно, то $\neg\sigma(a_1,...,a_n)$ доказуемо.

Рассмотрим теперь утверждение $n \in K$ (т. е. $\exists y \sigma_R(x,y)$), которое мы взяли в подразд.4.5 в качестве формального эквивалента утверждения $n \in K$. Тогда из леммы 4.5 вытекает следующее.

Следствие. Для любого натурального числа n из $n \in K$ следует, что утверждение $n \in K$ доказуемо в арифметике Пеано.

Доказательство. Пусть $n \in K$. Тогда существует натуральное число m такое, что $R(n,m)$ истинно, откуда по лемме 4.5 мы получаем, что утверждение $\sigma_R(n,m)$ доказуемо. Правила исчисления предикатов таковы, что мы имеем возможность немедленно найти доказательство для $\exists y \sigma_R(n,y)$; следовательно, утверждение $n \in K$ доказуемо.

В процессе доказательства Геделю потребовалось дополнительное техническое условие, которое носит название ω-непротиворечивости. Формальная система называется ω-непротиворечивой, если не существует утверждения $\tau(y)$ такого, что все следующие утверждения являются доказуемыми:

$$\exists y \tau(y),\ \neg\tau(0),\ \neg\tau(1),\ \neg\tau(2),\ ...$$

(ω-непротиворечивость – более сильное условие, чем условие непротиворечивости 4.2, б′.

В предположении ω-непротиворечивости из леммы 4.5 легко вывести утверждение, обратное приведенному выше следствию.

Лемма 4.6. Пусть арифметика Пеано ω-непротиворечива. Тогда для любого натурального числа n из доказуемости $n \in K$ следует, что $n \in K$.

Доказательство. Предположим, что $n \notin K$. Тогда для любого $m \in \square$ имеем $R(n,m)$ ложно, откуда по лемме 4.5 $\neg\sigma_R(n,m)$ доказуемо. Таким образом, если доказуемо $n \in K$, в то время как $n \notin K$, то доказуемы все следующие утверждения:

$$y\sigma_R(n,y),\quad \sigma_R(n,0),\quad \sigma_R(n,1),\ldots,$$

что противоречит ω-непротиворечивости утверждения $\tau(y)=\sigma_R(n,y)$.

Теперь мы можем доказать основную теорему.

Теорема 4.21 (теорема Геделя о неполноте [1931]). В языке L существует утверждение σ такое, что

а) если арифметика Пеано ω-непротиворечива, то σ недоказуемо;

б) если арифметика Пеано ω-непротиворечива, то $\neg\sigma$ недоказуемо.

Доказательство.

а) Напомним определение множеств, введенных выше:

$\mathrm{Pr}^* = \{n \mid n \in K$ доказуемо$\}$,

$\mathrm{Ref}^* = \{n \mid n \in K$ опровержимо$\}$.

Согласно следствию из леммы 4.5, $K \subseteq \mathrm{Pr}^*$, тогда как условие непротиворечивости дает, что $\mathrm{Pr}^* \cap \mathrm{Ref}^* = \varnothing$, откуда $\mathrm{Ref}^* \subseteq \bar{K}$. Будем рассуждать так же, как и в предыдущем варианте: множество Ref^* является рекурсивно перечислимым, возьмем поэтому такое m, что $\mathrm{Ref}^* = W_m$. Эта ситуация проиллюстрирована на рис. 4.3, который полезно сравнить с рис. 4.2.

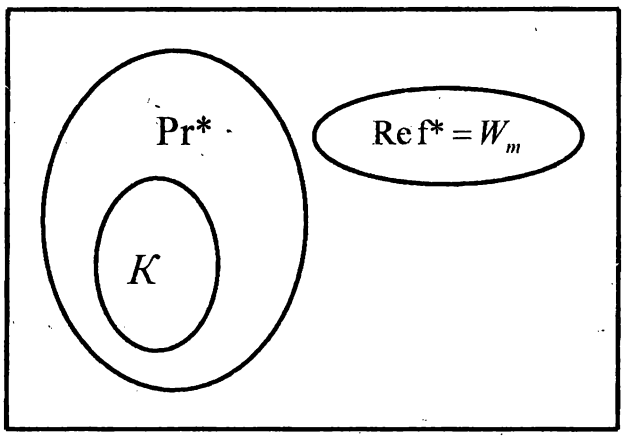


Рис. 4.3. Теорема Геделя о неполноте (теорема 4.21, п. а)

Из продуктивности $\bar{K}$ вытекает, что $m \in \bar{K} \setminus \mathrm{Ref}^*$; в частности, $m \notin \mathrm{Ref}^*$. Это означает, что утверждение $m \notin K$ недоказуемо.

Следовательно, чтобы доказать п. а, достаточно взять в качестве σ утверждение $m \notin K$.

б) Из условия ω-непротиворечивости следует (по лемме 4.6), что $\mathrm{Pr}^* \subseteq K$ и, следовательно, что $\mathrm{Pr}^* = K$. Таким образом, при условии ω-непротиворечивости ситуация рис. 4.3 преобразуется в ситуацию, изображенную на рис. 4.4.

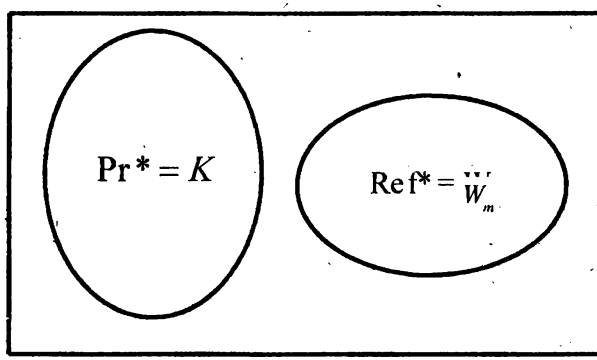


Рис. 4.4. Теорема Геделя о неполноте (теорема 4.21, п. б)

Тот факт, что $m \notin K$, означает, что $m \notin \mathrm{Pr}^*$, т. е. что утверждение $m \in K$ недоказуемо. Правила исчисления предикатов немедленно показывают, что утверждение $\neg\sigma$ (т.е. $m \notin K$) тоже недоказуемо.

Замечания.

1. Построенное в теореме Геделя утверждение σ называется неразрешимым или неразрешенным утверждением арифметики Пеано. Как мы отмечали ранее, σ имеет неформальный смысл: "я не являюсь доказуемым", и поэтому на интуитивном уровне это утверждение истинно.

2. Ясно, что теорема Геделя применима к любой рекурсивно аксиоматизированной формальной системе, в которой все разрешимые отношения представимы (в смысле леммы 4.5). В частности, это верно для всех систем, более сильных, чем арифметика Пеано. Тем самым мы не можем избавиться от неполноты путем добавления новых аксиом, например аксиомы "σ или $\neg\sigma$". Полученная формальная система будет содержать новое неразрешимое утверждение.

3. Неразрешимое утверждение σ можно явным образом построить из определения арифметики Пеано, поскольку, пользуясь этим определением, можно эффективно найти индекс m множества Ref^*. Этот конструктивный аспект теоремы Геделя есть следствие того факта, что множество K креативно. Внимательно проанализировав доказательство, можно убедиться в том, что если вместо множества K взять любое нерекурсивное рекурсивно перечислимое множество A, то удастся показать лишь существование неразрешимого утверждения.

В 1936 году Дж. Б. Россер обнаружил, как можно избавиться от условия ω-непротиворечивости в части б) теоремы Геделя. Этот улучшенный Россером вариант метода Геделя заключается в использовании вместо креативного множества K того факта, что рекурсивно перечислимые множества

$$K_0 = \{x : \phi_x(x) = 0\} \text{ и } K_1 = \{x : \phi_x(x) = 1\}$$

являются эффективно рекурсивно неотделимыми.

Тогда имеет место следующая лемма.

Лемма 4.7. В арифметике Пеано для любого числа n:

а) если $n \in K_0$, то утверждение $n \in K_0$ доказуемо;

б) если $n \in K_1$, то утверждение $n \in K_1$ доказуемо;

в) если доказуемо $n \in K_1$, то доказуемо и $n \notin K_0$.

Основной результат можно сформулировать в виде следующей теоремы.

Теорема 4.22 (теорема Геделя–Россера о неполноте). Существует утверждение τ такое, что если арифметика Пеано непротиворечива, то утверждения τ и $\neg\tau$ недоказуемы.

Замечания.

1. Утверждение $\neg\tau$, построенное при доказательстве этой теоремы, соответствует неразрешимому утверждению τ теоремы 4.21. Легко видеть, что $\neg\tau$ имеет неформальный смысл: "я не являюсь доказуемым" и тем самым интуитивно истинно.

2. Теорема Геделя–Россера применима к любой рекурсивно аксиоматизированной формальной системе арифметики, в которой все разрешимые отношения представимы и для которой может быть доказана лемма 4.7. Так что тем самым нет никакой возможности избавиться от неполноты, добавляя новые аксиомы.

Как мы уже видели, множество $\mathcal{T}$ (истинные утверждения

арифметики) не является рекурсивным (теорема 4.19). Это свойство часто обозначают другим термином, говоря, что множество $\mathcal{T}$ неразрешимо. И вообще, рассматривая множества утверждений, часто говорят "разрешимое" и "неразрешимое", имея в виду рекурсивное и нерекурсивное.

Для любой формальной системы арифметики мы можем задать вопрос: является ли множество $\mathcal{P}r$ доказуемых утверждений разрешимым? Ответ на него будет неизменно отрицателен, и убедиться в этом можно различными способами. Мы ограничимся одним из многих результатов в этой области, в доказательстве которого используются идеи, приведенные в начале этого подраздела.

Теорема 4.23. Пусть $(\mathcal{A}, \mathcal{D})$ – ω-непротиворечивая формальная система арифметики, в которой все разрешимые предикаты представимы (в смысле леммы 4.5). Тогда множество доказуемых утверждений креативно.

Следствие. Если арифметика Пеано ω-непротиворечива, то все доказуемые утверждения образуют креативное множество (это справедливо, в частности, в том случае, если все доказуемые утверждения истинны).

5. СЛОЖНОСТЬ ВЫЧИСЛЕНИЙ И ЭЛЕМЕНТЫ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ

5.1. Меры сложности вычислений

В общей теории алгоритмов изучается лишь принципиальная возможность алгоритмического решения задачи. При рассмотрении конкретных задач не обращается внимание на ресурсы времени и памяти для соответствующих им разрешающих алгоритмов. Однако ясно, что принципиальная возможность алгоритмического решения еще не означает, что оно может быть практически получено. Поэтому возникает потребность уточнить понятие алгоритмической разрешимости, введя характеристики слабости алгоритмов, позволяющие судить об их практической реализуемости. Ранее было установлено, что различные модели алгоритмов приводят к одному и тому же классу алгоритмически вычислимых функций. В то же время ясно, что выбор алгоритмической модели, реализующей алгоритмы, существенно влияет на сложность вычислений. Трудоемкость вычислений, вообще говоря, не сохраняется при изменении модели алгоритма. Однако имеется ряд фактов, которые

не зависят от вычислительной модели и относятся к так называемой машинно-независимой теории сложности.

Сложностью называется "трудность" решения вычислительных проблем, измеренная в терминах некоторого ресурса, потребляемого в процессе вычислений.

Анализ сложности вычислительных проблем в настоящее время является областью интенсивных исследований, в которых получаются важные практические результаты.

Вычислительные ресурсы (в теории алгоритмов) – это временные или пространственные характеристики процесса вычисления.

Вычислительный ресурс, потребляемый в процессе вычислений, может быть абстрактным или конкретным.

Критерий сложности – это средство измерения объема ресурсов, используемых в процессе вычисления.

Классы сложности – способ группировки алгоритмов или вычислимых функций в соответствии с их сложностью, при этом функции одного и того же класса имеют равную вычислительную сложность.

Классификации по сложности обычно подвергаются формальные языки, распознаваемые машиной Тьюринга.

Введем необходимые определения, выбрав машину Тьюринга в качестве модели вычислительного устройства. Пусть машина Тьюринга M вычисляет словарную функцию $f(x)$.

Определение 5.1. Временной сложностью называется функция $t_M(x)$, равная числу шагов машины M, выполненному при вычислении $f(x)$, если $f(x)$ определена. Если $f(x)$ не определена, то значение $t_M(x)$ считается неопределенным.

Определение 5.2. Активной зоной при работе машины M на слове x называется множество всех ячеек ленты, которые содержат непустые символы либо посещались в процессе вычисления $f(x)$ головкой машины M.

Определение 5.3. Пространственной сложностью (емкостной сложностью) называется функция $s_M(x)$, равная длине активной зоны при работе машины M на слове x, если $f(x)$ определена. Если $f(x)$ не определена, то значение $s_M(x)$ считается неопределенным.

Теорема 5.1. Пусть машина Тьюринга M имеет внешний и внутренний алфавиты мощности k и r соответственно. Тогда для функций сложности $t_M(x)$ и $s_M(x)$ справедливы оценки:

$$s_M(x) \leq |x| + t_M(x),$$
$$t_M(x) \leq rs_M^2(x)k^{s_M(x)}.$$

Доказательство. В начальный момент на ленте записано слово x длины $|x|$. На каждом шаге активными становятся не более одной новой ячейки, поэтому $s_M(x) \leq |x| + t_M(x)$.

Найдем теперь число различных конфигураций $K = a^{(1)}...a^{(i-1)}q_j a^{(i)}a^{s'}$, $s' \leq s$ с активной зоной s', не превосходящей фиксированной величины s. Имеется $k^{s'} \leq k^s$ вариантов записи на ленте, r вариантов выбора положения головки и s вариантов для значения длины s конфигурации K. Таким образом, число рассматриваемых конфигураций не превосходит rs^2k^s. Далее, если в процессе работы машины встретятся две одинаковые конфигурации, то машина зациклится, поскольку конфигурация однозначно определяет следующую за ней. Значит, если машина в процессе вычисления использует зону $s_M(x)$, то число ее шагов не превосходит числа различных конфигураций с зоной, не превышающей $s_M(x)$. В итоге получается неравенство $t_M(x) \leq rs_M^2(x)k^{s_M(x)}$.

Введенные функции сложности $t_M(x)$ и $s_M(x)$ являются словарными. Удобно ввести для рассмотрения функции натурального аргумента, которые также называются функциями временной и пространственной сложности.

Определение 5.4. **Временной ресурс** (в теории алгоритмов) – функция вида $Time(M,x)$, определяемая как число шагов в вычислении машиной Тьюринга M на входной цепочке символов x до останова M.

Определение 5.5. **Пространственный ресурс** (в теории алгоритмов) – это функция вида $Space(M,x)$, определяемая как число квадратов (полей) ленты (абстрактной) в вычислении машиной Тьюринга M на входной цепочке символов x.

Чтобы не путать пространственный ресурс, необходимый для работы машины Тьюринга, с пространственным ресурсом, выделяемым для входной цепочки x, предполагается, что машина Тьюринга имеет ленту ввода, которая работает только на считывание.

Определение 5.6. **Пространственная сложность вычисления *M*** – целочисленная функция $S_M(n)$, где $S_M(n) = \max\left(s_M(x) : |x| = n\right)$.

Определение 5.7. **Временная сложность вычисления *M*** – это целочисленная функция $T_M(n)$, где $T_M(n) = \max\left(t_M(x) : |x| = n\right)$.

Поведение этих функций в пределе при увеличении размера задачи n называется асимптотической временной (соответственно пространственной) сложностью. Для конкретных задач, как правило, находятся асимптотические функции сложности.

Для функций $t_M(x)$ и $s_M(x)$ справедливы свойства:

1) $D\left(t_T(x)\right) = D\left(f_T(x)\right)$, где $D\left(g(x)\right)$ – область определения функции $g(x)$, $f_T(x)$ – функция, вычисляемая машиной M;

2) предикат $P(x,y)$, определенный соотношением $P(x,y) \equiv \left(t_M(x) = y\right)$, разрешим. Аналогично для функции $s_M(x)$.

Дадим теперь определение абстрактной меры вычисления для одноместных числовых функций.

Определение 5.8. Пусть $f_1(x), f_2(x), \ldots f_n(x)$ – нумерация одноместных ч.р.ф. Мерой сложности вычислений называется любое семейство функций $\Phi_0(x), \Phi_1(x), \ldots, \Phi_n(x), \ldots$, обладающее свойствами:

1) $D\left(\Phi_i(x)\right) = D(f_i) \quad \forall i \in N_0$;

2) предикат $P_i(x;y) = \left(\Phi_i(x) = y\right)$ разрешим $\forall i \in N_0$.

5.2. Формальные языки класса *P*

Вопрос о верхней границе сложности вычислений формулируется так: существует ли такая о.р.ф. $h(x)$, что для любой вычислимой функции $f(x)$ найдется машина Тьюринга, для которой $t_M(x) \le h(x)$ (соответственно $s_M(x) \le h(x)$) там, где значения $t_M(x)$ определены. Ответ на этот вопрос отрицательный.

Если же говорить о нижних оценках сложности вычислений, то установление прямых нижних оценок удается лишь в очень редких случаях. В связи с этим получил распространение подход, связанный с получением косвенных нижних оценок, т.е. установление таких утверждений, в которых существование эффективного разрешающего алгоритма для конкретной задачи влечет за собой существование эффективного алгоритма для многих общепризнанно трудных задач.

Нам необходимо формализовать соответствующий подход. Пусть Π – некоторая массовая задача, характеризуемая множеством параметров, а $I \in \Pi$ – индивидуальная задача, в которой эти параметры фиксированы. Пусть с массовой задачей Π связана и зафиксирована система кодирования α, которая ставит каждой индивидуальной задаче $I \in \Pi$ в соответствие слово $\alpha(I)$ в некотором алфавите A. При этом под размером задачи I понимается длина слова $\alpha(I)$. Пусть M – машина Тьюринга, решающая задачу Π и $T_M(n) = \max\left(t_M(I) : |\alpha(I)| = n\right)$ – соответствующая функция временной сложности.

Определение 5.9. **Полиномиальное время** – характеристика сложности алгоритма, которая определяется как число элементарных операций, необходимых для реализации конкретного алгоритма обработки данных с длиной последовательности n, растущее с увеличением значений n менее быстро, чем полином $p(n)$.

Определение 5.10. Задача Π разрешима за полиномиальное время, если существует машина Тьюринга M, решающая ее за полиномиальное время.

Определение 5.11. **Формальные языки класса *P*** – класс формальных языков, которые считаются распознаваемыми за

полиномиальное время, точнее, язык L входит в P, если существует программа для машины Тьюринга M и некоторый полином $p(n)$, при котором эта программа распознает язык L, и для всех неотрицательных значений целых n соблюдается условие $T_M(n) \le p(n)$, где $T_M(n)$ является временной сложностью программы.

Класс языков с экспоненциально ограничивающими функциями представляет собой объединение бесконечного множества языков с полиномиальными ограничивающими функциями.

Алгоритм, сложность которого вырастает по экспоненте, называется экспоненциально ограниченным, при этом подразумевается, что алгоритм завершается.

Замечания.

1. В определении класса *P* существенным является фиксация схемы кодирования α. Многие естественные схемы кодирования полиномиально эквивалентны, т.е. позволяют переходить от одного кода задачи к другому коду за полиномиальное время от длины кода. В этом случае принадлежность задачи Π классу *P* определяется инвариантно по отношению к схемам кодирования. Однако это справедливо не всегда и, вообще говоря, класс сложности *P* зависит от схемы кодирования, поэтому там, где схема кодирования не очевидна или может повлиять на класс сложности, ее следует указывать явно.

2. Класс *P* определен через функцию временной сложности машины Тьюринга. Можно сделать соответствующие определения через любую другую алгоритмическую модель. Однако имеется ряд фактов о полиномиальной эквивалентности временных функций сложности многих типов вычислительных моделей, что позволяет утверждать, что класс *P* определен однозначно для "разумных" вычислительных моделей.

Необходимо отметить, что имеется существенное различие между алгоритмами полиномиальной и экспоненциальной сложности. Любой полиномиальный алгоритм более эффективен при достаточно больших размерах входа. Кроме того, полиномиальные алгоритмы лучше реагируют на рост производительности ЭВМ.

Полиномиальные алгоритмы обладают свойством замкнутости: их можно комбинировать, используя один в качестве "подпрограммы" другого и при этом результирующий алгоритм будет полиномиальным. В силу приведенных причин используется следующая терминология: полиномиальные алгоритмы называют эффективными, полиномиально

решаемые задачи называют легкорешаемыми, а экспоненциально решаемые задачи – труднорешаемыми. Для практики важным является классификация задач по признаку труднорешаемости, хотя установление легкорешаемости задачи еще не означает ее практическую решаемость.

5.3. Недетерминированная машина Тьюринга и язык *NP*

Введем понятие недетерминированной машины Тьюринга, схема которой приведена на рис. 5.1. **Недетерминированная машина Тьюринга** – это машина Тьюринга M, в которой на каждом этапе вычислений существуют альтернативы.

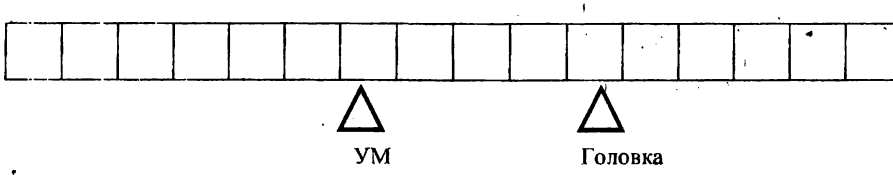


Рис. 5.1. Схема недетерминированной машины Тьюринга

191 Ее отличие от обычной машины Тьюринга заключается в том, что недетерминированная машина (НМ) имеет дополнительно угадывающий модуль (УМ), который может только записывать на ленту слова из внешнего алфавита. Поскольку речь идет о задачах распознавания, то удобно считать, что машина имеет два заключительных состояния q_Y и q_N, соответствующие ответам "ДА" и "НЕТ". Работа машины имеет две стадии.

Стадия 1 – угадывание. В начальный момент на ленте записано слово $x = x_1,...,x_n$ – код индивидуальной задачи I, начиная с ячейки 1. В ячейке 0 записан знак раздела *. Угадывающий модуль просматривает ячейку –1. Затем УМ пишет произвольное слово $c(x)$ по одной букве за такт в каждой ячейке с номерами –1, –2, –3,... . В итоге стадии 1 конфигурацией машины будет $c(x) * q_1 x$.

Стадия 2 – решение. На этой стадии недетерминированная машина работает как обычная машина Тьюринга в конфигурации $c(x)*q_1x$. Если машина через конечное число шагов приходит в состояние q_y, то говорят, что она принимает конфигурацию $c(x)*q_1x$. Будем говорить, что недетерминированная машина принимает x, если существует слово $c(x)$ такое, что принимается конфигурация $c(x)*q_1x$.

Классы сложности могут быть определены и для программ недетерминированной машины Тьюринга.

Определим время работы недетерминированной машины, положив

$$t_{HM}(x) = t_1(x) + t_2(x) \text{ (если } x \text{ принимается)},$$

где $t_1(x)$ – время работы на стадии 1, т.е., по определению, $t_1(x) = |c(x)|$;

$t_2(x)$ – время работы на стадии 2, т.е., по определению, $t_2(x) = t_M(c(x)*x)$;

M – соответствующая (обычная) машина Тьюринга.

Определим $T_{HM}(n) = \max\left(t_{HM}(x) : |x| = n\right)$.

Недетерминированная машина НТ решает задачу Π за полиномиальное время, если

$$t_{HM}(n) = O\left(p(n)\right)$$

для некоторого полинома p.

Заметим, что временная функция определена только для тех индивидуальных задач x, которые принимаются машиной НТ.

Определение 5.12. **Класс NP языков** – класс формальных языков, которые считаются распознаваемыми за полиномиальное время недетерминированной машиной Тьюринга.

Рассмотрим теперь вопрос о взаимоотношении между введенными классами *P* и *NP*. Ясно, что $P \subseteq NP$. Имеется много причин считать это включение строгим, однако этот факт пока не доказан.

Теорема 5.2. Если задача $\Pi \in NP$, то существует такой полином p, что Π может быть решена детерминированным алгоритмом со сложностью $O\left(2^{p(n)}\right)$, – размер Π.

Доказательство. Пусть НТ – недетерминированная машина Тьюринга, решающая задачу Π, и $q(n)$ – полином, ограничивающий временную функцию сложности НТ. Не нарушая общности, можно считать, что $q(n) = c_1 n^{c_2}$ (c_1, c_2 – константы). По определению класса *NP*, если вход x длины n принимается НТ, то существует слово $c(x)$ длины не более чем $q(n)$ такое, что НТ дает ответ "ДА" не более чем за $q(n)$ шагов. Значит, общее число возможных слов-догадок не более чем $k^{q(n)}$, где k – мощность внешнего алфавита НТ. Можно считать, что все догадки имеют длину $q(n)$, в противном случае их можно подравнять. Теперь можно представить детерминированный алгоритм решения задачи Π, который на каждом из $k^{q(n)}$ слов-догадок реализует стадию 2 работы НТ и работает $q(n)$ тактов. Алгоритм дает ответ "ДА", если найдется слово-догадка, приводящее к принимающему вычислению. Время работы данного алгоритма $q(n) \cdot k^{q(n)}$. Ясно, что существует подходящий полином $p(n)$ и сложность описанного алгоритма не превосходит $O\left(2^{p(n)}\right)$.

Замечания.

1. Класс *NP* – один и тот же для различных вычислительных моделей, использующих недетерминированные операции.

2. Класс *P* замкнут относительно дополнения задач. Для класса *NP* этого утверждать нельзя. Дополнением $\overline{A}$ задачи распознавания A называют задачу, в которой коды задач с ответом "ДА" в точности соответствуют кодам задач A, которые не имеют ответ "ДА". Класс задач, являющихся дополнениями к задачам класса *NP*, обозначают *CO – NP*.

5.4. Понятие *NP*-полной задачи

Если $P \neq NP$, то задачи из $NP \setminus P$ являются трудноразрешимыми. Цель дальнейших рассуждений состоит в доказательстве того, что существуют конкретные задачи Π, для которых справедливо включение $\Pi \in NP \setminus P$, если $P \neq NP$. Соответствующие результаты основаны на понятии полиномиальной сводимости задач. Пусть Π_1, Π_2 – две задачи

распознавания, задаваемые в алфавитах A_1 и A_2 соответственно. Будем говорить, что задача Π_1 полиномиально сводится к задаче Π_2 (обозначение $\Pi_1 \propto \Pi_2$), если существует словарная функция $f: A_1^* \to A_2^*$ такая, что выполнены условия:

1) f – полиномиально вычислима;

2) $\forall x \in A_1^*$ выполнено, x – индивидуальная $\Leftrightarrow f(x) \in A_2^*$ – индивидуальная задача Π_1 с ответом "ДА" и задача Π_2 с ответом "ДА".

Теорема 5.3. Если выполнено $\Pi_1 \propto \Pi_2$ и $\Pi_2 \in P$, то $\Pi_1 \in P$.

Доказательство. Предположим полиномиальный алгоритм решения задачи $\Pi_1 : x \in \Pi_1$, найдем $f(x) \in \Pi_2$ и применим к $f(x)$ полиномиальный алгоритм. Если получен ответ "ДА", то x имеет ответ "ДА". В противном случае x имеет ответ "НЕТ". Время работы алгоритма не превосходит $p_f(|x|) + p_2(p_f(|x|))$, где p_f – полином, ограничивающий время вычисления функции f, участвующей в сведении $\Pi_1 \propto \Pi_2$; p_2 – полином, ограничивающий время решения задачи Π_2. Теорема доказана.

Теорема 5.4. Если выполнено $\Pi_1 \propto \Pi_2$ и $\Pi_2 \propto \Pi_3$, то $\Pi_1 \propto \Pi_3$.

Доказательство очевидно, так как функция $f_3(x) = f_2(f_1(x))$ осуществляет сведение $\Pi_1 \propto \Pi_3$, если f_1 и f_2 дают сведения $\Pi_1 \propto \Pi_2$ и $\Pi_2 \propto \Pi_3$ соответственно.

Определение 5.13. Задача Π называется NP-полной, если выполнены следующие условия:

а) $\Pi \in NP$

б) $\Pi_1 \propto \Pi$ для любой задачи $\Pi_1 \in NP$.

Класс NP-полных задач обозначается через NPC.

Задача Π называется NP-трудной, если для нее выполнено условие б). Класс NP-трудных задач обозначается через NPH.

Согласно определению справедливы следующие соотношения:

$NPC \cap P \neq \varnothing \Rightarrow P = NP$;

$NPH \cap P \neq \varnothing \Rightarrow P = NP$;

$NPC \cap (NP \setminus P) \neq \varnothing \Rightarrow NPC \cap P = \varnothing$.

Другими словами, если для какой-нибудь NP-полной задачи существует полиномиально разрешающий алгоритм, то для любой задачи из класса NP существует полиномиально разрешающий алгоритм. То же высказывание справедливо относительно NP-трудной задачи. Если какая-либо NP-полная задача не лежит в классе P, то и все NP-полные задачи не лежат в классе P.

Теорема 5.5. Если задачи $\Pi_1, \Pi_2 \in NP$, $\Pi_1 \propto \Pi_2$ и $\Pi_1 \in NPC$, то $\Pi_2 \in NPC$.

Доказательство. Пусть $\Pi' \in NP$ – произвольная задача. Тогда по определению $\Pi' \propto \Pi_1$. Поскольку по условию $\Pi_1 \propto \Pi_2$, то согласно теореме 5.4 получается $\Pi' \propto \Pi_2$, что и доказывает теорему.

Отсюда получается способ доказательства NP-полноты конкретной задачи путем полиномиального сведения к ней другой NP-полной задачи. Этот же метод пригоден и для доказательства NP-трудности задачи.

5.5. Хорновские дизъюнкты

Математическая логика является теоретической основой логического программирования. Здесь мы вернемся к определению понятий математической логики, начатому в подразд. 1.4.

Среди множества всевозможных предложений особое значение имеют предложения, содержащие связки "и" ($\wedge$) и одну связку "если" ($\leftarrow$). Такие предложения называются дизъюнктами. Дадим более точное определение.

Определение 5.14. Дизъюнкт – это предложение вида

$$A_1 \wedge A_2 \wedge ... \wedge A_k \leftarrow B_1 \wedge B_2 \wedge ... \wedge B_n , \qquad (*)$$

где $A_1, A_2, ..., A_k, B_1, B_2 ... B_n$ – предикаты или литеры.

Особое значение для логического программирования имеет понятие дизъюнкта Хорна.

Определение 5.15. *Дизъюнктом Хорна* называется такой дизъюнкт, у которого $k = 0$ или $k = 1$. При $k = n = 0$ получается пустой дизъюнкт, обозначаемый символом Л.

Если $k = 0$, то из формулы (*) получается дизъюнкт вида

$$B_1 \wedge B_2 \wedge ... \wedge B_n .$$

Этот дизъюнкт называется вопросом.

Среди множества всех дизъюнктов особую роль играют те, для которых $k = 1$. Если при этом еще и $n = 0$, то такой дизъюнкт называется фактом: A1.

Если же $n > 0$, дизъюнкт называется правилом

$$A \leftarrow B_1 \wedge B_2 \wedge ... \wedge B_n ,$$

где A – голова, а $B_1, B_2, ..., B_n$ – цели, образующие тело правила.

Необходимо отметить свойство формулы (*). Если воспользоваться, приведенными выше формулами сокращения, связывающими логические связки, то оказывается, что любой дизъюнкт можно преобразовать к такой формуле, что в записи ее будут использованы только связки $\vee$ (или), $\neg$ (не).

Введенные определения позволяют установить следующую иерархию:

- формула;
- предложение;
- дизъюнкт;
- дизъюнкт Хорна;
- факт, правило, вопрос, пустой дизъюнкт.

Базой знаний будет называться всякое (непустое) множество фактов и правил. Записанная в базе знаний информация представлена в некоторой формально-логической системе и предназначена для изучения информации посредством применения определенного процесса рассуждений, базирующегося на логическом выводе. Эти рассуждения основаны на понятии логического следствия. Известно, что логическое следствие из заданной совокупности допущений или предположений истинно, если истинны эти предположения. Однако одна

и та же формула может быть истинной или ложной в зависимости от ее содержательного значения. Например, логическая формула

$$(x \succ y) \leftarrow (|x| \succ |y|)$$

(знак $x \succ y$ означает отношение "*x* больше *y*", а знак $|x|$ – абсолютную величину *x*) истинна, если *x* и *y* принадлежат множеству положительных чисел, и ложна, если *x* и *y* принадлежат множеству отрицательных чисел. В математической логике существует понятие интерпретации, а в разных интерпретациях одна и та же формула может принимать разные значения.

Для интерпретации множества предложений *S* выбирается множество объектов *D*, называемое областью интерпретации.

Интерпретация множества предложений *S* над *D* состоит из следующих трех соответствий:

1. Каждой константе (числовой или литеральной) ставится в соответствие некоторый элемент из *D*.

2. Каждому *n*-арному функтору из *S* сопоставляется отображение n-ки из D_n в *D*.

3. Каждому *n*-арному предикатному символу из *S* сопоставляется отображение n-ки из D_n в множество {ИСТИНА, ЛОЖЬ}.

Примером интерпретации для множества предложений, описывающих родственные отношения:

{мама(Саша, Оля), мама(Петя, Маша), сын(*x*, *y*) ← мама(*y*, *x*)}

являются следующие правила.

1. Область интерпретации *D* = {Саша, Оля, Петя, Маша}.

2. Функторы отсутствуют.

3. Отображение:

а) мама(*x*, *y*) → {ИСТИНА, ЛОЖЬ},

б) сын(*x*, *y*) → {ИСТИНА, ЛОЖЬ}.

Точное определение логического следствия основано на понятии выполнимости формулы (предложения). Формула (предложение) выполняется в интерпретации над множеством *D* тогда и только тогда, когда это предложение в данной интерпретации принимает значение ИСТИНА. Соответственно множество предложений *S* выполняется в данной интерпретации тогда и только тогда, когда выполняется любое предложение из *S*. *S* логически влечет некоторое предложение *s* тогда и только тогда, когда для любой интерпретации над любой областью *D* из выполнимости множества *S* следует выполнимость в ней *s*. Если вернуться к последнему примеру, то множество предложений:

S = {мама(Саша, Оля), мама(Петя, Маша), сын(x, y) ← мама(y, x)}

логически влечет s = сын(Маша, Петя). Это обозначается $S \models s$.

Смысл отношения $\models$ состоит в том, что если $S \models s$, то это происходит из-за внутренней структуры предложений множества S, независимо от того, что могут обозначать составляющие их части в той или иной интерпретации. Использование интерпретаций – способ получения понятия логического следствия, которое, в конечном счете, не зависит от этих интерпретаций.

Связь логического программирования и логического следствия определяется тем способом, которым задачи формулируются и решаются. Основная задача, стоящая перед пользователем при логическом программировании, состоит в том, чтобы описать условия задачи и критерии ее решения. Таким образом, логическая сущность задачи в этом случае становится более ясной. Существуют методы построения логического следствия, не зависящие от интерпретации. Одним из таких методов является метод резолюций. Он основан на следующем заключении.

Пусть S – некоторое множество дизъюнктов и U – дизъюнкт. Необходимо определить, является ли U логическим следствием из S. Для этого множество S дополняется отрицанием U. Если окажется, что объединение множеств S и $\{U\}$ противоречиво, то U логически следует из S.

Предположим, что даны два дизъюнкта C_1 и C_2 из S, а L_1 и L_2 – комплементарные литеры из C_1 и C_2 соответственно (литеры U и $\bar{U}$ называются комплементарными). Дизъюнкция C_1 и C_2 $(C_1 \wedge C_2)$, в которой предварительно вычеркнуты литеры L_1 и L_2, называется резольвентой.

Принцип резолюции основан на том, что резольвента дизъюнктов C_1 и C_2 является логическим следствием C_1 и C_2. Если в результате многократного применения процесса построения резольвент получается пустой дизъюнкт, то множество дизъюнктов S противоречиво. Более того, для противоречивого множества дизъюнктов применениями процесса построения резольвент всегда можно получить пустой дизъюнкт (Л).

В процедуре доказательства по методу резолюций для того, чтобы отождествить комплементарные пары литер, необходимо унифицировать выражения, иными словами, необходимо найти подстановку, которая

может сделать несколько выражений тождественными.

Унификацией двух выражений E_1 и E_2 с переменными $v_1;...,v_n$ (или без них) называется нахождение такой подстановки (может быть, и пустой), что выражения E_1 и E_2 совпадут. Такая подстановка называется унификатором. Если оказалось, что для данных термов не существует унификатора, то термы не унифицируемы.

5.6. Логические программы

Все языки программирования можно разделить на процедурные и декларативные. Подавляющее большинство используемых в настоящее время языков программирования (Си, Паскаль, Бейсик и т. п.) относятся к процедурным языкам. Наиболее существенными классами декларативных языков являются функциональные (Лисп, Лого, АПЛ и т. п.) и логические (Пролог, Плэнер, Конивер и др.) языки.

На практике языки программирования не являются чисто процедурными, функциональными или логическими, а содержат в себе черты языков различных типов. На процедурном языке часто можно написать функциональную программу или ее часть и наоборот. Возможно, было бы точнее вместо типа языка говорить о стиле или методе программирования. Естественно, различные языки поддерживают разные стили в разной степени.

Процедурная программа состоит из последовательности операторов и предложений, управляющих последовательностью их выполнения. Типичными операторами являются операторы присваивания и передачи управления, операторы ввода-вывода и специальные предложения для организации циклов. Из них можно составлять фрагменты программ и подпрограммы. В основе процедурного программирования лежат взятие значения какой-то переменной, совершение над ним действия и сохранение нового значения с помощью оператора присваивания, и так до тех пор, пока не будет получено (и, возможно, напечатано) желаемое окончательное значение.

Логическое программирование – это один из подходов к информатике, при котором в качестве языка высокого уровня используется логика предикатов первого порядка в форме фраз Хорна. Логика предикатов первого порядка – это универсальный абстрактный язык, предназначенный для представления знаний и для решения задач. Его можно рассматривать как общую теорию отношений. Логическое программирование базируется на подмножестве логики

предикатов первого порядка, при этом оно одинаково широко с ней по сфере охвата. Логическое программирование дает возможность программисту описывать ситуацию при помощи формул логики предикатов, а затем для выполнения выводов из этих формул применить автоматический решатель задач (т. е. некоторую процедуру). При использовании языка логического программирования основное внимание уделяется описанию структуры прикладной задачи, а не выработке предписаний компьютеру о том, что ему следует делать. Другие понятия информатики из таких областей, как теория реляционных баз данных, программная инженерия и представление знаний, также можно описать (и, следовательно, реализовать) с помощью логических программ.

Функциональная программа состоит из совокупности определений функций. Функции, в свою очередь, представляют собой вызовы других функций и предложений, управляющих последовательностью вызовов. Вычисления начинаются с вызова некоторой функции, которая в свою очередь вызывает функции, входящие в ее определение и т. д. в соответствии с иерархией определений и структурой условных предложений. Функции часто либо прямо, либо опосредованно вызывают сами себя.

Каждый вызов возвращает некоторое значение в вызывавшую его функцию, вычисление которой после этого продолжается; этот процесс повторяется до тех пор, пока запустившая вычисления функция не вернет конечный результат пользователю.

"Чистое" функциональное программирование не признает присваиваний и передач управления. Разветвление вычислений основано на механизме обработки аргументов условного предложения. Повторные вычисления осуществляются через рекурсию, являющуюся основным средством функционального программирования.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Составьте логическую схему базы знаний.

2. Докажите следующее утверждение арифметики Пеано: если $x \prec y$ и $y \prec z$, то $x \prec z$.

3. Вычислите геделевский номер следующих п.р.ф.:

$$f(x, y) = 2(x, y);$$

$$f(x, y, z, t) = e_4^3.$$

4. Расшифруйте натуральное число $n = 37$.

$e_1^1 - 5$ $e_2^1 - 7$ $e_2^2 - 11$

$e_3^1 - 13$ $e_3^2 - 17$ $e_3^3 - 23$

$e_4^1 -$ $e_4^2 -$. $e_4^4 - 37$

5. С помощью таблицы истинности докажите следующее утверждение:

$$p \supset q = \neg p \vee q.$$

p	q	$p \supset q$	$\neg q$	$\neg p \vee q$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	1	0	1

6. Найдите функцию $h(m,k)$ в рекуррентной формуле $f(n+1) = h\big(n, f(n)\big)$ для одноместной функции $f(n)$:

$$f(n) = 2 \cdot 4 \cdot \ldots \cdot (2n-2) \cdot 2n.$$

7. Найдите функции g и h в рекурсивной формуле для двухместной функции $f(x,y)$, если рекурсия проводится: 1) по переменной x; 2) по переменной y:

$$f(x,y) = y \cdot 2^x$$

8. Найдите функции g и h в рекурсивной формуле для трехместной функции $f(x,y,z)$, если рекурсия проводится по переменной y:

$$f(x,y,z)=(x+z)\cdot y.$$

9. Найдите рекурсивные формулы для двухместной функции $f(x,y)$, если рекурсия проводится по x:

$$f(x,y)=(x+y)^2.$$

10. Найдите рекурсивные формулы для трехместной функции $f(x,y,z)$, если рекурсия проводится по x:

$$f(x,y,z)=(x+z)\cdot y.$$

11. Докажите, что двухместная функция $f(x,y)$ принадлежит классу п.р.ф., если рекурсия проводится по y:

$$f(x,y) = 2x + y + 2.$$

ТРЕНИНГ УМЕНИЙ

1. Пример выполнения упражнения тренинга на умение № 1.

Задание

Найдите функцию $h(m,k)$ в рекуррентной формуле $f(n+1)=h\big(n,f(n)\big)$ для одноместной функции $f(n)=\sin \pi n$.

Решение

Заполните таблицу, подобрав к каждому алгоритму конкретное соответствие из задания.

№ п/п	Алгоритм	Конкретное соответствие данной ситуации предложенному алгоритму
1.	Подставить в функцию $f(n)$ вместо n значение $n+1$	$f(n+1)=\sin\big(\pi(n+1)\big)=$ $=\sin(\pi n+\pi)=-\sin(\pi n)$
2.	Выразить $f(n+1)$ через $f(n)$ и n	$f(n+1)=-f(n)$
3.	Найти функцию $h(m,k)$ из формулы $f(n+1)=h\big(f(n),n\big)$, заменив $f(n)$ на m, а затем n – на k	$h(m,k)=-m$

Решите самостоятельно:

Найдите функцию $h(m,k)$ в рекуррентной формуле $f(n+1)=h\big(n,f(n)\big)$ для одноместной функции $f(n)$:

1.1. $f(n)=n^2$

1. $g(0)=f(0)=0^2=0$
2. $f(n+1)=(n+1)^2=n^2+2n+1=f(n)+2n+1$

$f(n)=m,\ n=k$

$h(m,k)=m+2k+1$

✓ 1.2. $f(n)=\frac{1}{3^n}$

$g(0)=f(0)=\frac{1}{3^0}=\frac{1}{1}=1$

$f(n+1)=\frac{1}{3^{n+1}}=\frac{1}{3^n\cdot 3}=\frac{1}{3^n}\cdot\frac{1}{3}=f(n)\cdot\frac{1}{3}$

$f(n)=m,\ n=$

$h(m)=\frac{1}{3}m$

✓ 1.3. $f(n)=(2n)!$

$g(0)=f(0)=(2\cdot 0)!=0$

$f(n+1)=(2(n+1))!=(2n+2)!=(2n)!(2n+1)(2n+2)$

$f(n)=m,\ n$

$h(m)=$

✓ 1.4. $f(n)=1\cdot 3\cdot\ldots\cdot(2n-1)\cdot(2n+1)$

$g(0)=f(0)=$

$f(n+1)=1\cdot 3\ldots(2n+1)(2n+3)=f(n)(2n+3)$

$m\ 2k+3$

1.5. $f(n)=\sum_{k=1}^{n}k$

1. $g(0)=f(0)=0$

2) $f(n+1)=\sum_{k=1}^{n+1}k=\sum_{k=1}^{n}k+(n+1)=f(n)+n+1$

$f(n)=m,\ n=k$

$h(m,k)=m+k+1.$

2. Пример выполнения упражнения тренинга на умение № 2.

Задание

Найдите функции g и h в рекурсивной формуле для двухместной функции $f(x,y)=x\cdot y$, если рекурсия проводится по переменной x.

Решение

Заполните таблицу, подобрав к каждому алгоритму конкретное соответствие из задания.

№ п/п	Алгоритм	Конкретное соответствие данной ситуации предложенному алгоритму
1.	Найти функцию g (см. замечание): $g(y)=f(0,y)$ (так как рекурсия проводится по x)	$g(y)=f(0,y)=0$
2.	Подставить в функцию $f(x,y)$ вместо x значение $x+1$	$f(x+1,y)=(x+1)y=xy+y$
3.	Выразить функцию $f(x+1;y)$ через $f(x,y)$, x, y	$f(x+1,y)=f(x,y)+y$
4.	Найти функцию $h(x,y,z)$ из формулы $f(x+1,y)=h\big(x,y,f(x,y)\big)$, заменив $f(x,y)$ на z	$h(x,y,z)=z+y$

Решите самостоятельно:

Найдите функции g и h в рекурсивной формуле для двухместной функции f, если рекурсия проводится по x:

2.1. $f(x,y)=x^2+y$

$g(y)=f(0,y)=0^2+y=y$

$f(x+1,y)=(x+1)^2+y=x^2+2x+1+y=x^2+y+2x+1=$

$=f(x,y)+2x+1$

$f(x,y)=z$

$h(x,y,z)=z+2x+1$

2.2. $f(x,t) = x^2 + t^2$

$g(t) = f(0,t) = 0^2 + t^2 = t^2$

$f(x+1, t) = (x+1)^2 + t^2 = x^2 + 2x + 1 + t^2 = x^2 + t^2 + 2x + 1 =$

$= f(x,t) + 2x + 1$

$f(x,y) = z$

$h(x,y,z) = z + 2x + 1$

2.3. $f(t,x) = (t+x)^2$

Найдите функции g и h в рекурсивной формуле для двухместной функции f, если рекурсия проводится по y:

2.4. $f(x,y) = 3^{x+y}$

$g(x) = f(x,0) = 3^x$

$f(x,y+1) = 3^{x+y+1} = 3^{x+y} \cdot 3 = f(x,y) \cdot 3$

$f(x,y) = z$

$h(x,y,z) = 3z.$

2.5. $f(x,y) = x^2 + y$

3. Пример выполнения упражнения тренинга на умение № 3.

Задание

Найдите функции g и h в рекурсивной формуле для трехместной функции $f(x,y,z) = x \cdot y + z$, если рекурсия проводится по переменной y.

Решение

Заполните таблицу, подобрав к каждому алгоритму конкретное соответствие из задания.

№ п/п	Алгоритм	Конкретное соответствие данной ситуации предложенному алгоритму
1.	Найти функцию g: $g(x,z) = f(x,0,z)$ (так как рекурсия проводится по y)	$g(x,z) = f(x,0,z) = z$
2.	Подставить в функцию $f(x,y,z)$ вместо y значение $y+1$	$f(x,y+1,z) = x(y+1)+z = xy+z+x$
3.	Выразить функцию $f(x,y,z+1)$ через $f(x,y,z), x, y, z$	$f(x,y+1,z) = (xy+z)+x = f(x,y,z)+x$
4.	Найти функцию $h(x,y,z,t)$ из формулы $f(x,y,z+1) = h\big(x,y,z,f(x,y,z)\big)$, заменив $f(x,y,z)$ на t	$h(x,y,z,t) = t+x$

Решите самостоятельно:

Найдите функции g и h в рекурсивной формуле для двухместной функции f, если рекурсия проводится по x:

3.1. $f(x,y,z) = xy^2z$

$g(0,y,z) = f(0,y,z) = 0$

$f(x+1,y,z) = f(x,y,z) + y^2z + z$

$f(x,y,z) = t$

$f(t) = t + y^2z + z$

3.2. $f(x,y,z) = x^2(y+z)$

Найдите функции g и h в рекурсивной формуле для двухместной функции f, если рекурсия проводится по y:

3.3. $f(x,y,z) = x^2y + y^2z$

Найдите функции g и h в рекурсивной формуле для двухместной функции f, если рекурсия проводится по z:

3.4. $f(x,y,z) = xy + z$

3.5. $f(x,y,z) = 2^{x+y+z}$

4. Пример выполнения упражнения тренинга на умение № 4.

Задание

Найдите рекурсивные формулы для двухместной функции $f(x, y) = xy$, если рекурсия проводится по x.

Решение

Заполните таблицу, подобрав к каждому алгоритму конкретное соответствие из задания.

№ п/п	Алгоритм	Конкретное соответствие данной ситуации предложенному алгоритму
1.	С помощью умения 2 найти функции g и h	$g(y) = 0$, $h(x, y, z) = z + y$
2.	Записать рекурсивные формулы, используя соотношения $\begin{cases} f(0, y) = g(y) \\ f(x+1, y) = h(x, y, z)\big\|_{z=f(x,y)} \end{cases}$ (так как рекурсия проводится по x)	$\begin{cases} 0 \cdot y = 0 \\ (x+1)y = xy + y \end{cases}$

Решите самостоятельно:

Найдите рекурсивные формулы для двухместной функции $f(x, y)$, если рекурсия проводится по x:

4.1. $f(x, y) = x^2 + y$

4.2. $f(x, t) = x^2 + t^2$

4.3. $f(t,x)=(t+x)^2$

Найдите рекурсивные формулы для двухместной функции $f(x,y)$, если рекурсия проводится по y:

4.4. $f(x,y)=3^{x+y}$

4.5. $f(x,y)=x^2+y$

5. Пример выполнения упражнения тренинга на умение № 5.

Задание

Найдите рекурсивные формулы для трехместной функции $f(x,y,z)=xy+z$, если рекурсия проводится по y.

Решение

Заполните таблицу, подобрав к каждому алгоритму конкретное соответствие из задания.

№ п/п	Алгоритм	Конкретное соответствие данной ситуации предложенному алгоритму
1.	С помощью умения 3 найти функции g и h	$g(x,z)=z$, $h(x,y,z,t)=t+x$
2.	Записать рекурсивные формулы, используя соотношения $\begin{cases} f(x,0,z)=g(x,z) \\ f(x,y+1,z)=h(x,y,z,t)\big\|_{t=f(x,y,z)} \end{cases}$ (так как рекурсия проводится по y)	$\begin{cases} x\cdot 0+z=z \\ x(y+1)+z=(xy+z)+x \end{cases}$

Решите самостоятельно:

Найдите рекурсивные формулы для трехместной функции f, если рекурсия проводится по x:

5.1. $f(x,y,z)=xy^2z$

5.2. $f(x,y,z)=x^2(y+z)$

Найдите рекурсивные формулы для трехместной функции f, если рекурсия проводится по y:

5.3. $f(x, y, z) = x^2 y + y^2 z$

Найдите рекурсивные формулы для трехместной функции f, если рекурсия проводится по z:

5.4. $f(x, y, z) = xy + z$

5.5. $f(x, y, z) = 2^{x+y+z}$

6. Пример выполнения упражнения тренинга на умение № 6.

Задание

Докажите, что двухместная функция $f(x, y) = xy$ принадлежит классу п.р.ф., если рекурсия проводится по x.

Решение

Заполните таблицу, подобрав к каждому алгоритму конкретное соответствие из задания.

№ п/п	Алгоритм	Конкретное соответствие данной ситуации предложенному алгоритму
1.	С помощью умения 2 найти функции g и h	$g(y)=0$, $h(x,y,z)=z+y$
2.	Выразить $g(x)$ через исходные п.р.ф. (так как рекурсия проводится по x)	$g(x)=0=0(x)$
3.	Выразить $h(x,y,z)$ через исходные п.р.ф.	$h(x,y,z)=z+y=e_3^2(x,y,z)+e_3^3(x,y,z)$
4.	Сделать вывод о принадлежности функции классу п.р.ф.: если функция $f(x,y)$ может быть получена из исходных п.р.ф с помощью операторов подстановки и рекурсии, то она принадлежит классу п.р.ф.	Функция $f(x,y)=xy$ может быть получена из исходных п.р.ф. с помощью операторов рекурсии, следовательно, она принадлежит классу п.р.ф.

Решите самостоятельно:

Докажите, что двухместная функция f принадлежит классу п.р.ф., если рекурсия проводится по x:

6.1. $f(x,y)=x+y$

6.2. $f(x,t)=2xt$

6.3. $f(t,x) = t + 3x$

Докажите, что двухместная функция f принадлежит классу п.р.ф., если рекурсия проводится по y:

6.4. $f(x,y) = xy + y + x$

6.5. $f(x,y) = x + 2y + 1$

7. Пример выполнения упражнения тренинга на умение № 7.

Задание

Найдите хотя бы один геделевский номер ng одноместной функции $f(x) = x + 2$.

Решение

Заполните таблицу, подобрав к каждому алгоритму конкретное соответствие из задания.

№ п/п	Алгоритм	Конкретное соответствие данной ситуации предложенному алгоритму
1.	Найти число c по формуле $c = g(0) = f(0)$	$c = g(0) = 2$
2.	Подставить в функцию $f(x)$ вместо x значение $x+1$	$f(x+1) = (x+1)+2$
3.	Выразить функцию $f(x+1)$ через $f(x)$ и x	$f(x+1) = f(x)+1$
4.	Найти функцию $h(x,y)$ из формулы $f(x+1) = h(x, f(x))$, заменив $f(x)$ на y	$h(x,y) = y+1 = S(y)$
5.	Найти гёделевский номер j функции h	$j = 3$
6.	Найти гёделевский номер ng функции $f(x)$ по формуле $ng = 2^{c+1} \cdot 5^{j}$, где $c = g(0)$	$ng(x+1) = 2^{2+1} \cdot 5^3 = 2^3 \cdot 5^3 = 1000$

Решите самостоятельно:

Найдите хотя бы один гёделевский номер ng одноместной функции $f(x)$:

7.1. $f(x) = x$

1) $g(0) = f(0) = 0$

2) $f(x+1) = x+1 = f(x)+1$

$f(x) = m$

$h(m, k) = m+1 = S(m$

3) номер: $2^{c+1} \cdot 5^{j}$, $c = g(0)$

4) $0(x) \to 2 \Rightarrow c = 2$

$S(m) \to 3 \Rightarrow j = 3$

$2^{2+1} \cdot 5^3 = 2^3 \cdot 5^3 = 10^3 = 1000$

$f(x)$ имеет номер 1 ее 1000.

7.2. $f(x) = x+3$

1) $c = g(0) = 3$

$f(x+1) = (x+1)+2$

$f(x+1) = f(x)+1$

$h(x,y) = y+1 = S(y)$

$j = 3$

$ng(x+1) = 3^{2+1} \cdot 5^3 = 3^3 \cdot 5^3 = 1000$

7.3. $f(x) = x + n$

7.4. $f(x) = 1(x)$

7.5. $f(x) = 3(x)$

8. Пример выполнения упражнения тренинга на умение № 8.

Задание

Найдите хотя бы один геделевский номер ng двухместной функции $f(x, y) = x + y$.

Решение

Заполните таблицу, подобрав к каждому алгоритму конкретное соответствие из задания.

№ п/п	Алгоритм	Конкретное соответствие данной ситуации предложенному алгоритму
1.	Найти функцию $g(x) = f(x, 0)$	$g(x) = x$
2.	Найти функцию $h(x, y, z)$ из уравнения $f(x, y+1) = h(x, y, f(x, y))$ (см. умение 2)	$f(x, y+1) = x + y + 1 = f(x, y) + 1$ $h(x, y, z) = z + 1 = S(z)$
3.	Найти геделевский номер i функции $g(x)$ (см. умение 7)	$g(x) = x = e_1^1 \Rightarrow i = 5$

4.	Найти геделевский номер j функции $h(x,y,z)$	$h(x,y,z)=S(z)\Rightarrow j=3$
5.	Найти геделевский номер ng двухместной функции $f(x,y)$ по формуле $ng\left(f(x,y)\right)=3^{f}\cdot 5^{f}$	$ng(x+y)=3^{5}\cdot 5^{3}$

Решите самостоятельно:

Найдите хотя бы один гёделевский номер ng двухместной функции $f(x,y)$:

8.1. $f(x,y)=xy$

8.2. $f(x,y)=(x+1)y$

8.3. $f(x,y)=x+y+1$

8.4. $f(x,y)=0(x,y)$

8.5. $f(x,y)=1(x,y)$

9. Пример выполнения упражнения тренинга на умение № 9.

Задание

Найдите хотя бы один геделевский номер ng трехместной функции $f(x,y,z)=x+y+z$.

Решение

Заполните таблицу, подобрав к каждому алгоритму конкретное соответствие из задания.

№ п/п	Алгоритм	Конкретное соответствие данной ситуации предложенному алгоритму
1.	Найти функцию $g(x,y)=f(x,y,0)$	$g(x,y)=x+y$
2.	Найти функцию $h(x,y,z,t)$ из уравнения $f(x,y,z+1)=h\big(x,y,z,f(x,y,z)\big)$ (см. умение 2)	$f(x,y,z+1)=x+y+z+1=f(x,y,z)+1$ $h(x,y,z,t)=t+1=S(t)$
3.	Найти геделевский номер i функции $g(x,y)$ (см. умение 7)	$g(x,y)\Rightarrow i=3^5\cdot 5^3$ (см. пример тренинга 8)

4.	Найти геделевский номер j функции $h(x,y,z,t)$	$h(x,y,z,t)=S(t)\Rightarrow j=3$
5.	Найти геделевский номер ng двухместной функции $f(x,y,z)$ по формуле $ng\left(f(x,y,z)\right)=3^i\cdot 5^j$	$ng(x+y+z)=3^{3^3\cdot 5^1}\cdot 5^3$

Решите самостоятельно:

Найдите хотя бы один гёделевский номер ng трехместной функции $f(x,y,z)$:

9.1. $f(x,y,z)=x+y+z+1$

9.2. $f(x,y,z)=x+y+z+2$

9.3. $f(x,y,z)=0(x,y,z)$

9.4. $f(x,y,z)=1(x,y,z)$

9.5. $f(x,y,z)=e_5^2(x,y,z)$

10. Пример выполнения упражнения тренинга на умение № 10.

Задание

Расшифруйте натуральное число $n=500$.

Решение

Заполните таблицу, подобрав к каждому алгоритму конкретное соответствие из задания.

№ п/п	Алгоритм	Конкретное соответствие данной ситуации предложенному алгоритму
1.	Если число n простое, то $f'=e_k'$	$n=500$ - не простое
2.	Если число $n=2^{c+1}\cdot 5^j$, то: если j – номер какой-либо двухместной п.р.ф., то n – номер одноместной п.р.ф, получающейся из c и h с помощью оператора рекурсии; если j – не номер п.р.ф, то n – не номер п.р.ф.	$n=2^2\cdot 5^3$ $c=1,\ f(0)=1$ $j=3,\ h(x,y)=S(y)=y+1$ $f(x)=x+1$

3.	Если $n \neq 2^{k+1} \cdot 5^{j}$, то проверяем, равно ли $n = 3^{i} \cdot 5^{j}$; если да, то i и j – номера п.р.ф. вместимости $k-1$ и $k+1$ соответственно и т.д.	

Решите самостоятельно:

Расшифруйте натуральные числа:

10.1. $n = 23$

10.2. $n = 250$

10.3. $n = 225$

10.4. $n = 4000$

10.5. $n = 3^{7} \cdot 5^{37}$

Система Пеано

Наиболее употребительная система аксиом арифметики натуральных чисел, предложенная в конце XIX в. независимо немецким математиком *Р.* Дедекиндом и итальянским математиком Дж. Пеано, известна под названием системы Пеано. Неопределяемые объекты этой системы – натуральное число (в дальнейшем мы для краткости будем говорить просто "число") и нуль. Ее единственное неопределяемое отношение – "число *y* следует за числом *x*′′(обозначение: $y = x'$). Аксиом всего пять:

1. Нуль есть число.
2. Для всякого числа x существует единственное число *y*, такое, что $y = x'$.
3. Если $x' = y'$, то $x = y$.
4. Нуль не следует ни за каким числом.
5 (аксиома индукции). Если *B* – произвольное свойство чисел, справедливое для нуля и такое, что, каково бы ни было число *x*, справедливости свойства *B* для *x* следует справедливость его для x', то свойство *B* справедливо для всех чисел.

Приведем некоторые утверждения арифметики.

Утверждение 1. Любое число или есть нуль, или следует за каким-либо числом.

Утверждение 2. Существует единственная функция двух переменных, сопоставляющая каждым двум числам *x*, *y* число $x + y$ таким образом, что выполняются условия:

$$x + 0 = x;$$
$$x + y' = (x + y)'.$$

Утверждение 3. $(x + y) + M = x + (y + M)$.

Утверждение 4. $0 + x = x$.

Утверждение 5. $x + y = y + x$.

Утверждение 6. Существует единственная функция двух переменных, сопоставляющая каждым двум числам *x*, *y* число $x \cdot y$ таким образом, что выполняются условия:

$$x \cdot 0 = 0;$$
$$x \cdot y' = x \cdot y + x.$$

Утверждение 7. $(x + y) \cdot z = x \cdot z + y \cdot z$.

Утверждение 8. $0 \cdot x = 0$.

Утверждение 9. $x \cdot y = y \cdot x$.

Утверждение 10. $x \cdot (y + z) = x \cdot y + x \cdot z$.

Утверждение 11. $(x \cdot y) \cdot z = x \cdot (y \cdot z)$.

Утверждение 12. Невозможно $x \prec x$.

Утверждение 13. Если $x \prec y$ и $y \prec z$, то $x \prec z$.

Изд. лиц. ЛР № 071765 от 07.12.1998
Казахстанско-Российский университет
Объем 12 п.л. Тираж 577.

15/8/5

Изд. лиц. ЛР № 071765 от 07.12.1998
Казахстанско-Российский университет
Объем 12 п.л. Тираж 577.

75701